STACS 2008, 21-23 February 2008

Proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science

Bordeaux, France, 2008

Edited by: Susanne Albers and Pascal Weil

2008

Published by: IBFI Schloss Dagstuhl
Printed by: Imprimerie de l'Université Bordeaux I
ISBN: 978-3-939897-06-4

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 1-6
www.stacs-conf.org

FOREWORD

SUZANNE ALBERS [1] AND PASCAL WEIL [2]

[1] Institut für Informatik, Universität Freiburg
E-mail address: salbers@informatik.uni-freiburg.de

[2] LaBRI, Université de Bordeaux, France
E-mail address: pascal.weil@labri.fr

The Symposium on Theoretical Aspects of Computer Science (STACS) is held alternately in France and in Germany. The conference of February 21-23, 2008, held in Bordeaux, is the 25th 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), Montpellier (2004), Stuttgart (2005), Marseille (2006) and Aachen (2007).

The interest in STACS has remained at a high level over the past years. The STACS 2008 call for papers led to approximately 200 submissions from 38 countries. Each was assigned to at least three program committee members. The program committee held a 2-week long electronic meeting at the end of November, to select 54 papers. As co-chairs of this committee, we would like to sincerely thank its members and the many external referees for the valuable work they put into the reviewing process. The overall very high quality of the papers that were submitted to the conference made this selection a difficult task.

We would like to express our thanks to the three invited speakers, Maxime Crochemore, Thomas Schwentick and Mihalis Yannakakis, for their contributions to the proceedings.

Special thanks are due to A. Voronkov for his EasyChair software (www.easychair.org) which gives the organisers of conferences such as STACS a remarkable level of comfort; to Ralf Klasing for helping us explore the many possibilities of this brilliant software; to Emilka Bojańczyk for the design of the STACS poster, proceedings and logo; and to the members of the Organizing Committee, chaired by David Janin.

An innovation in this year's STACS is the electronic format of the publication. A printed version was also available at the conference, with ISBN 978-3-939897-06-4. The electronic proceedings are available through several portals, and in particular through HAL and DROPS. HAL is an electronic repository managed by several French research agencies, and DROPS is the Dagstuhl Research Online Publication Server. We want to thank both these servers for hosting the proceedings of STACS and guaranteeing them perennial availability. The rights on the articles in the proceedings are kept with the authors and the papers are available freely, under a Creative Commons license (see www.stacs-conf.org/faq.html for more details).

Conference organization

Members of the Program Committee

Manindra Agrawal, *IIT Kanpur*
Susanne Albers, *Freiburg University*, co-chair
Danièle Beauquier, *Paris-12 University, Créteil*
Mikołaj Bojańczyk, *Warsaw University*
Nadia Creignou, *Marseille-3 University*
Anna Gàl, *University of Texas, Austin*
Naveen Garg, *IIT Delhi*
Kazuo Iwama, *Kyoto University*
Juhani Karhumäki, *Turku University*
Hartmut Klauck, *Frankfurt University*
Kamal Lodaya, *IMSc, Chennai*
Christof Löding, *RWTH, Aachen*
Frédéric Magniez, *Paris-11 University, Orsay*
Peter Bro Miltersen, *Aarhus University*
Vahab Mirrokni, *Microsoft Research, Redmond*
Seth Pettie, *University of Michigan, Ann Arbor*
Eric Rivals, *CNRS and Montpellier University*
Nicole Schweikardt, *TU Berlin*
Christian Sohler, *Paderborn University*
Howard Straubing, *Boston College*
Klaus Wagner, *Würzburg University*
Pascal Weil, *CNRS and Bordeaux University*, co-chair

Members of the Organizing Committee

Véronique Bogati
Bruno Courcelle
David Janin (Local chair)
Mamadou Kante
Ralf Klasing
Olivier Ly
Frédéric Mazoit
Anca Muscholl
Géraud Sénizergues
Igor Walukiewicz
Pascal Weil
Marc Zeitoun

External reviewers

Scott Aaronson
Mohammad Ali Abam
Sarmad Abbasi
Luca Aceto
Heiner Ackermann
Marcel Ackermann
Isolde Adler
Pavan Aduri
Tatsuya Akutsu
Laurent Alonso
Helmut Alt
Andris Ambainis
Amihood Amir
Reid Andersen
Daniel Andersson
Takahito Aoto
S. Arun-Kumar
V. Arvind
Arash Asadpour
Eugène Asarin
Yossi Azar
Mohsen Bayati
Srečko Brlek
Amitabha Bagchi
Nikhil Bansal
Vince Bárány
Régis Barbanchon
David Barrington
Surender Baswana
Niel de Beaudrap
Mark De Berg
Vincent Berry
Yves Bertot
Alexis Bès
Stephane Bessy
Olaf Beyersdorff
Jean-Camille Birget
Henrik Björklund
Achim Blumensath
Nicolas Bonichon
Glencora Borradaile
Yacine Boufkhad
Mathilde Bouvel
Andreas Brandstädt
Marcus Brazil
Dirk Brendel
Harry Buhrman
Wojciech Buszkowski
Jin-Yi Cai
Cezar Câmpeanu
Arnaud Carayol
Olivier Carton
Jorge Castro
Annie Chateau
Jingchao Chen
Yijia Chen
Victor Chepoi
Giorgos Christodoulou
Joëlle Cohen
Dave Cohen
David Cohen-Steiner
Thomas Colcombet
Éric Colin de Verdière
Robert Cori
Gérard Cornuéjols
Bruno Courcelle
Maxime Crochemore
Artur Czumaj
Deepak D'Souza
Ovidiu Daescu
Hervé Daudé
Jean-Paul Delahaye
Gilles Didier
Catalin Dima
Dominique Barth
Michael Drmota
Stefan Droste
Ioana Dumitriu
Jacques Duparc
Bruno Durand
Herbert Edelsbrunner
Uwe Egly
Patricia Evans
Jean-Claude Fournier
Rolf Fagerberg
Jittat Fakcharoenphol
Stephen Fenner
Thomas Fernique
Matthias Fischer
Fedor Fomin
Christiane Frougny
Hiroshi Fujiwara
Péter Gács
Travis Gagie
Vijay Garg
Cyril Gavoille
Hugo Gimbert
Rodolphe Giroudeau
Christian Glaßer
Daniel Gonçalves
Daniel Gottesman
Chris Gray
Fred Green
Irène Guessarian
Anupam Gupta
Michel Habib
Mohammad HajiAghayi
Vesa Halava
Yijie Han
Xin Han
Tero Harju
Ishay Haviv
André Hernich
Mika Hirvensalo
John Hitchcock
Juha Honkala
Hendrik Jan Hoogeboom
Joseph Horton
Peter Høyer
Szczepan Hummel
Ferran Hurtado
Oscar Ibarra
Keiko Imai
Nicole Immorlica
Hiro Ito
Gábor Ivanyos
Alain Jean-Marie
David Jacobs
Sanjay Jain
David Janin
Jesper Jansson
T.S. Jayram
Peter Jeavons
Fan Jianxi
Pushkar Joglekar
Satyen Kale
Mark Kambites
Makoto Kanazawa
Jarkko Kari
Wong Karianto
Akinori Kawachi
Neeraj Kayal
Julia Kempe
Michael Kerber

Iordanis Kerenidis
Majid Khabbazian
Daniel Kirsten
Felix Klaedtke
Adam Klivans
Johannes Köbler
Jochen Könemann
Eryk Kopczyński
Frédéric Koriche
Artur Korniłowicz
Guy Kortsarz
Michal Koucký
Stephan Kreutzer
Marc van Kreveld
Andrei Krokhin
Manfred Kufleitner
Ravi Kumar
Michal Kunc
Manfred Kunde
Piyush Kurur
Dietrich Kuske
Tomi Kärki
Markku Laine
Christiane Lammersen
Sophie Laplante
Sławomir Lasota
Aurélien Lemay
Leonid Levin
Leonid Libkin
Nutan Limaye
Markus Lohrey
Sylvain Lombardy
Zvi Lotker
Martin Lotz
Jack Lutz
P. Madhusudan
Veli Makinen
Johann Makowsky
Andreas Malcher
Amal Dev Manuel
Marc Kaplan
Brian Marcus
Wim Martens
Dániel Marx
Marios Mavronicolas
Elvira Mayordomo
Frédéric Mazoit
Andrew McGregor
Pierre McKenzie
Ingmar Meinecke
Guy Melançon
Wolfgang Merkle
Antoine Meyer
Friedhelm Meyer auf der Heide
Shuichi Miyazaki
Morteza Monemizadeh
Malika More
Marcin Mucha
Madhavan Mukund
Filip Murlak
Anca Muscholl
Rahul Muthu
S. Muthukrishnan
Phuong Nguyen
Arfst Nickelsen
François Nicolas
Joachim Nieren
Harumichi Nishimura
Damian Niwiński
Richard Nock
Gustav Nordh
Zeev Nutov
Jan Obdržálek
Yoshio Okamoto
Alexander Okhotin
Nicolas Ollinger
Hirotaka Ono
Friedrich Otto
Jérôme Palaysi
Konstantinos Panagiotou
Paritosh Pandya
Paweł Parys
Mihai Pătraşcu
Christophe Paul
Soumya Paul
Elisabeth Pelz
Mati Pentus
Sylvain Perifel
Ion Petre
Ulrich Pferschy
Fabrice Philippe
Jean-Éric Pin
Marcus Pivato
Alexander Rabinovich
Harald Räcke
Prasad Raghavendra
M. Sohel Rahman
Venkatesh Raman
R. Ramanujam
Rudy Raymond
Christian Reitwießner
Eric Rémila
Jochen Renz
Christian Retoré
Pierre-Alain Reynier
Liam Roditty
Heiko Röglin
Lajos Rónyai
Jörg Rothe
Michał Rutkowski
Kalle Saari
Kunihiko Sadakane
Mohammad Safari
Lakhdar Sais
Mohammad Reza Salavatipour
Alex Samorodnitsky
Peter Sanders
Miklos Santha
Rahul Santhanam
Luigi Santocanale
Jayalal Sarma
Srinivasa Rao Satti
Thomas Sauerwald
Saket Saurabh
Nitin Saxena
Nicolas Schabanel
Guido Schäfer
Gilles Schaeffer
Manfred Schmidt-Schauß
Lutz Schröder
Patrice Séébold
Luc Segoufin
Helmut Seidl
Victor Selivanov
Olivier Serre
Micha Sharir
Somnath Sikdar
Sunil Simon
Cristina Sirangelo
Anastasias Sidiropoulos
Michiel Smid
Troels Bjerre Sørensen
Gregory Sorkin
Robert Špalek
Alex Spelten
Magnus Steinby
Bernd Sturmfels
S.P. Suresh
Maxim Sviridenko
Chaitanya Swamy
Antonios Symvonis
Laurent Tichit
Yasuhiko Takenaga

Suguru Tamaki
Hisao Tamaki
Seiichiro Tani
Orestis Telelis
Kavitha Telikepalli
Pascal Tesson
Edouard Thiel
Thomas Thierauf
Wolfgang Thomas
Sebastien Tixeuil
Takeshi Tokuyama
Szymon Toruńczyk
Hélène Touzet
Mathieu Tracol
William Trotter
Mario Valencia-Pabon
Leslie Valiant
Niko Välimäki
Kasturi Varadarajan
Yann Vaxès
Nikolai Vereshchagin
Kumar Neeraj Verma
Daniel Vilenchik
Aymeric Vincent
Berthold Vöcking
Heribert Vollmer
Jan Vondrák
Daria Walukiewicz
Ingo Wegener
Udi Wieder
Prudence Wong
Camille Wormser
Masaki Yamamoto
Shigeru Yamashita
Koichi Yamazaki
Hiroki Yanagisawa
Raphael Yuster
Mariette Yvinec
Stathis Zachos
Konrad Zdanowski
Norbert Zeh
Sandra Zilles
Alexander Zvonkin
Paweł Żyliński

Table of contents

Foreword 1
S. Albers and P. Weil

Program Committee 2

External reviewers 3

Table of contents 7

Invited papers

Understanding Maximal Repetitions in Strings 11
M. Crochemore and L. Ilie

A Little Bit Infinite? On Adding Data to Finitely Labelled Structures . 17
T. Schwentick

Equilibria, Fixed Points and Complexity Classes 19
M. Yannakakis

Contributed papers

Pushdown Compression 39
P. Albert, E. Mayordomo, P. Moser and S. Perifel

Quantum search with variable times 49
A. Ambainis

Structural aspects of tilings 61
A. Ballier, B. Durand and E. Jeandel

Limit complexities revisited 73
L. Bienvenu, A. Muchnik, A. Shen and N. Vereshchagin

Trimmed Moebius Inversion and Graphs of Bounded Degree 85
A. Björklund, T. Husfeldt, P. Kaski and M. Koivisto

On the Complexity of the Interlace Polynomial 97
M. Bläser and C. Hoffmann

Minimizing Flow Time in the Wireless Gathering Problem 109
V. Bonifaci, P. Korteweg, A. Marchetti-Spaccamela and L. Stougie

On Termination for Faulty Channel Machines 121
P. Bouyer, N. Markey, J. Ouaknine, P. Schnoebelen and J. Worrell

Stackelberg Network Pricing Games 133
P. Briest, M. Hoefer and P. Krysta

Sublinear Communication Protocols for Multi-Party Pointer Jumping and a Related Lower Bound 145
J. Brody and A. Chakrabarti

Finding Irrefutable Certificates for S_2^p via Arthur and Merlin 157
V. Chakaravarthy and S. Roy

Quantifying Homology Classes 169
C. Chen and D. Freedman

Shortest Vertex-Disjoint Two-Face Paths in Planar Graphs 181
É. Colin de Verdière and A. Schrijver

Geodesic Fréchet Distance Inside a Simple Polygon 193
A. F. Cook IV and C. Wenk

Improved Algorithms for the Range Next Value Problem and Applications 205
M. Crochemore, C. Iliopoulos, M. Kubica, M. S. Rahman and T. Waleń

Connecting Polygonizations via Stretches and Twangs 217
M. Damian, R. Flatland, J. O'Rourke and S. Ramaswami

Deterministically Isolating a Perfect Matching in Bipartite Planar Graphs 229
S. Datta, R. Kulkarni and S. Roy

Tight Bounds for Blind Search on the Integers 241
M. Dietzfelbinger, J. E. Rowe, I. Wegener and P. Woelfel

Discrete Jordan Curve Theorem: A Proof Formalized in Coq with Hypermaps 253
J.-F. Dufourd

Trimming of Graphs, with Application to Point Labeling 265
T. Erlebach, T. Hagerup, K. Jansen, M. Minzlaff and A. Wolff

Computing Minimum Spanning Trees with Uncertainty 277
T. Erlebach, M. Hoffmann, D. Krizanc, M. Mihal'ák and R. Raman

Convergence Thresholds of Newton's Method for Monotone Polynomial Equations 289
J. Esparza, S. Kiefer and M. Luttenberger

Model Checking Games for the Quantitative mu-Calculus 301
D. Fischer, E. Grädel and L. Kaiser

Order-Invariant MSO is Stronger than Counting MSO in the Finite 313
T. Ganzow and S. Rubin

Succinctness of the Complement and Intersection of Regular Expressions 325
W. Gelade and F. Neven

Efficient Algorithms for Membership in Boolean Hierarchies of Regular Languages 337
C. Glaßer, H. Schmitz and V. Selivanov

On the Complexity of Elementary Modal Logics 349
E. Hemaspaandra and H. Schnoor

Fixed Parameter Polynomial Time Algorithms for Maximum Agreement and Compatible Supertrees 361
V. T. Hoang and W.-K. Sung

Complexity of solutions of equations over sets of natural numbers 373
A. Jeż and A. Okhotin

Cardinality and counting quantifiers on omega-automatic structures 385
L. Kaiser, S. Rubin and V. Bárány

On the Induced Matching Problem 397
I. Kanj, M. J. Pelsmajer, M. Schaefer and G. Xia

On Geometric Spanners of Euclidean and Unit Disk Graphs 409
I. Kanj and L. Perković

The Frobenius Problem in a Free Monoid 421
J.-Y. Kao, J. Shallit and Z. Xu

Space Hierarchy Results for Randomized Models 433
J. Kinne and D. van Melkebeek

Ehrenfeucht-Fraïssé Goes Automatic for Real Addition 445
F. Klaedtke

New Combinatorial Complete One-Way Functions 457
A. Kojevnikov and S. Nikolenko

Compatibility of Shelah and Stupp's and Muchnik's iterations with fragments of monadic second order logic 467
D. Kuske

Geometric Set Cover and Hitting Sets for Polytopes in $\mathbb{R}^3$ 469
S. Laue

A Theory for Valiant's Matchcircuits 491
A. Li and M. Xia

Rent, Lease or Buy: Randomized Algorithms for Multislope Ski Rental . 503
Z. Lotker, B. Patt-Shamir and D. Rawitz

Lower bounds for adaptive linearity tests 515
S. Lovett

An Improved Randomized Truthful Mechanism for Scheduling Unrelated Machines 527
P. Lu and C. Yu

Lagrangian Relaxation and Partial Cover 539
J. Mestre

On Dynamic Breadth-First Search in External-Memory 551
U. Meyer

Analytic aspects of the shuffle product 561
M. Mishna and M. Zabrocki

Weak index versus Borel rank 573
F. Murlak

A Mahler's theorem for functions from words to integers 585
J.-É. Pin and P. Silva

Distinguishing Short Quantum Computations 597
B. Rosgen

Factoring Polynomials over Finite Fields using Balance Test 609
C. Saha

On the decomposition of k-valued rational relations 621
J. Sakarovitch and R. de Souza

The Isomorphism Problem for Planar 3-Connected Graphs is in Unambiguous Logspace 633
T. Thierauf and F. Wagner

Efficient Minimization of DFAs with Partial Transition Functions 645
A. Valmari and P. Lehtinen

Design by Measure and Conquer - A Faster Exact Algorithm for Dominating Set 657
J. van Rooij and H. Bodlaender

Weighted Matching in the Semi-Streaming Model 669
M. Zelke

Author Index 681

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 11-16
www.stacs-conf.org

UNDERSTANDING MAXIMAL REPETITIONS IN STRINGS

MAXIME CROCHEMORE [1] AND LUCIAN ILIE [2]

[1] King's College London, Strand, London WC2R 2LS, United Kingdom
and Institut Gaspard-Monge, Université Paris-Est, France
E-mail address: maxime.crochemore@kcl.ac.uk

[2] Department of Computer Science, University of Western Ontario
N6A 5B7, London, Ontario, Canada
E-mail address: ilie@csd.uwo.ca

Abstract. The cornerstone of any algorithm computing all repetitions in a string of length n in $\mathcal{O}(n)$ time is the fact that the number of runs (or maximal repetitions) is $\mathcal{O}(n)$. We give a simple proof of this result. As a consequence of our approach, the stronger result concerning the linearity of the sum of exponents of all runs follows easily.

1. Introduction

Repetitions in strings constitute one of the most fundamental areas of string combinatorics with very important applications to text algorithms, data compression, or analysis of biological sequences. One of the most important problems in this area was finding an algorithm for computing all repetitions in linear time. A major obstacle was encoding all repetitions in linear space because there can be $\Theta(n \log n)$ occurrences of squares in a string of length n (see [1]). All repetitions are encoded in runs (that is, maximal repetitions) and Main [9] used the s-factorization of Crochemore [1] to give a linear-time algorithm for finding all leftmost occurrences of runs. What was essentially missing to have a linear-time algorithm for computing all repetitions, was proving that there are at most linearly many runs in a string. Iliopoulos et al. [4] showed that this property is true for Fibonacci words. The general result was achieved by Kolpakov and Kucherov [7] who gave a linear-time algorithm for locating all runs in [6].

Kolpakov and Kucherov proved that the number of runs in a string of length n is at most cn but could not provide any value for the constant c. Recently, Rytter [10] proved that $c \leq 5$. The conjecture in [7] is that $c = 1$ for binary alphabets, as supported by

1998 ACM Subject Classification: F.2.2 Nonnumerical Algorithms and Problems; G.2.1 Combinatorics.
Key words and phrases: combinatorics on words, repetitions in strings, runs, maximal repetitions, maximal periodicities, sum of exponents.
This work has been done during the second author's stay at Institut Gaspard-Monge. The same author's research was supported in part by NSERC.

computations for string lengths up to 31. Using the technique of this note, we have proved [2] that it is smaller than 1.6, which is the best value so far.

Both proofs in [6] and [10] are very intricate and our contribution is a simple proof of the linearity. On the one hand, the search for a simple proof is motivated by the very importance of the result – this is the core of the analysis of any optimal algorithm computing all repetitions in strings. None of the above-mentioned proofs can be included in a textbook. We believe that the simple proof shows very clearly why the number of runs is linear. On the other hand, a better understanding of the structure of runs could pave the way for simpler linear-time algorithms for finding all repetitions. For the algorithm of [6] (and [9]), relatively complicated and space-consuming data structures are needed, such as suffix trees.

The technical contribution of the paper is based on the notion of δ-close runs (runs having close centers), which is an improvement on the notion of neighbors (runs having close starting positions) introduced by Rytter [10].

On top of that, our approach enables us to derive easily the stronger result concerning the linearity of the sum of exponents of all runs of a string. Clearly this result implies the first one, but the converse is not obvious. The second result was given another long proof in [7]; it follows also from [10].

Finally, we strongly believe that our ideas in this paper can be further refined to improve significantly the upper bound on the number of runs, if not to prove the conjecture. The latest refinements and computations (December 2007) show a $1.084n$ bound.

2. Definitions

Let A be an alphabet and A^* the set of all finite strings over A. We denote by $|w|$ the length of a string w, by $w[i]$ its ith letter, and by $w[i\,..\,j]$ its factor $w[i]w[i+1]\cdots w[j]$. We say that w has period p iff $w[i] = w[i+p]$, for all $1 \le i \le |w| - p$. The smallest period of w is called *the period* of w and the ratio between the length and the period of w is called the *exponent* of w.

For a positive integer n, the nth power of w is defined inductively by $w^1 = w$, $w^n = w^{n-1}w$. A string is *primitive* if it cannot be written as a proper integer (two or more) power of another string. Any nonempty string can be uniquely written as an integer power of a primitive string, called its *primitive root*. It can also be uniquely written in the form $u^e v$ where $|u|$ is its (smallest) period, e is the integral part of its exponent, and v is a proper prefix of u.

The following well-known *synchronization* property will be useful: If w is primitive, then w appears as a factor of ww only as a prefix and as a suffix (not in-between). Another property we use is *Fine and Wilf's periodicity lemma*: If w has periods p and q and $|w| \ge p+q$, then w has also period $\gcd(p,q)$. (This is a bit weaker than the original lemma which works as soon as $|w| \ge p + q - \gcd(p,q)$, but it is good enough for our purpose.) We refer the reader to [8] for all concepts used here.

For a string $w = w[1\,..\,n]$, a *run*[1] (or maximal repetition) is an interval $[i\,..\,j]$, $1 \le i < j \le n$, such that (i) the factor $w[i\,..\,j]$ is periodic (its exponent is 2 at least) and (ii) both $w[i-1\,..\,j]$ and $w[i\,..\,j+1]$, if defined, have a strictly higher (smallest) period. As an example, consider $w = \mathsf{abbababbaba}$; $[3\,..\,7]$ is a run with period 2 and exponent 2.5; we have $w[3\,..\,7] = \mathsf{babab} = (\mathsf{ba})^{2.5}$. Other runs are $[2\,..\,3], [7\,..\,8], [8\,..\,11], [5\,..\,10]$ and $[1\,..\,11]$.

[1]Runs were introduced in [9] under the name *maximal periodicities*; the are called *m-repetitions* in [7] and *runs* in [4].

For a run starting at i and having period $|x| = p$, we shall call $w[i\,..\,i+2p-1] = x^2$ the *square* of the run (this is the only part of a run we can count on). Note that x is primitive and the square of a run cannot be extended to the left (with the same period) but may be extendable to the right. The *center* of the run is the position $c = i + p$. We shall denote the *beginning* of the run by $i_x = i$, the *end of its square* by $e_x = i_x + 2p - 1$, and its *center* by $c_x = i_x + p$.

3. Linear number of runs

We describe in this section our proof of the linear number of runs. The idea is to partition the runs by grouping together those having close centers and similar periods. To this aim, for any $\delta > 0$, we say that two runs having squares x^2 and y^2 are δ-*close* if (i) $|c_x - c_y| \leq \delta$ and (ii) $2\delta \leq |x|, |y| \leq 3\delta$. We prove that there cannot be more than three mutually δ-close runs. (There is one exception to this rule – case (vi) below – but then, even fewer runs are obtained.) This means that the number of runs with the periods between 2δ and 3δ in a string of length n is at most $\frac{3n}{\delta}$. Summing up for values $\delta_i = \frac{1}{2}\big(\frac{3}{2}\big)^i$, $i \geq 0$, all periods are considered and we obtain that the number of runs is at most

$$\sum_{i=0}^{\infty} \frac{3n}{\delta_i} = \sum_{i=0}^{\infty} \frac{3n}{\frac{1}{2}(\frac{3}{2})^i} = 18n. \tag{3.1}$$

For this purpose, we start investigating what happens when three runs in a string w are δ-close. Let us denote their squares by x^2, y^2, z^2, their periods by $|x| = p$, $|y| = q$, $|z| = r$, and assume $p \leq q \leq r$. We discuss below all the ways in which x^2 and y^2 can be positioned relative to each other and see that long factors of both runs have small periods which z^2 has to synchronize. This will restrict the beginning of z^2 to only one choice as otherwise some run would be left extendable. Then a fourth run δ-close to the previous three cannot exist.

Notice that, for cases (i)-(v) we assume the centers of the runs are different; the case when they coincide is covered by (vi).

(i) $(i_y < i_x <)c_y < c_x < e_x \leq e_y$. Then x and the suffix of length $e_y - c_x$ of y have period $q - p$; see Fig. 1(i). We may assume the string corresponding to this period is a primitive string as otherwise we can make the same reasoning with its primitive root.

Since z^2 is δ-close to both x^2 and y^2, it must be that $c_z \in [c_x - \delta\,..\,c_y + \delta]$. Consider the interval of length $q - p$ that ends at the leftmost possible position for c_z, that is, $I = [c_x - \delta - (q-p)\,..\,c_x - \delta - 1]$. It is included in the first period of z^2, that is, $[i_z\,..\,c_z - 1]$, and in $[i_x\,..\,c_y]$. Thus $w[I]$ is primitive and equal, due to z^2, to $w[I + r]$ which is a factor of $w[c_x\,..\,e_y]$. Therefore, the periods inside the former must synchronize with the ones in the latter. It follows, in the case $i_z > i_x - (q - p)$, that $w[i_z - 1] = w[c_z - 1]$, that is, z^2 is left extendable, a contradiction. If $i_z < i_x - (q-p)$, then $w[c_x - 1] = w[i_x - (q-p) - 1] = w[i_x - 1]$, that is, x^2 is left extendable, a contradiction. The only possibility is that $i_z = i_x - (q-p)$ and r equals q plus a multiple of $q - p$. Here is an example: $w = \mathsf{baababababaababababaab}$, $x^2 = w[5\,..\,14] = (\mathsf{ababa})^2$, $y^2 = w[1\,..\,14] = (\mathsf{baababa})^2$, and $z^2 = w[3\,..\,20] = (\mathsf{abababaab})^2$.

We have already, due to z^2, that $x = \rho^\ell \rho'$, where $|\rho| = q - p$ and ρ' a prefix of ρ. A fourth run δ-close to the previous three would have to have the same beginning as z^2 and the length of its period would have to be also q plus a multiple of $q - p$. This would imply an equation of the form $\rho^m \rho' = \rho' \rho^m$ and then ρ and ρ' are powers of the same string, a contradiction with the primitivity of x.

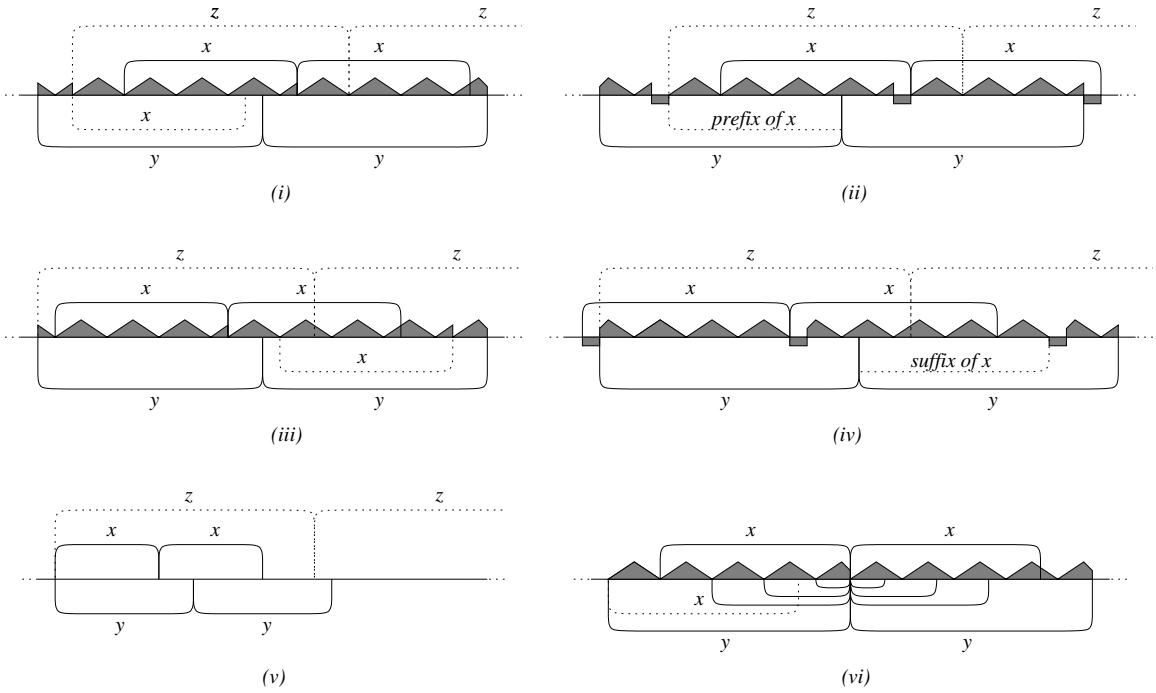


Figure 1: Relative position of x^2 and y^2.

(ii) $(i_y < i_x <)c_y < c_x < e_y \leq e_x$; this is similar with (i); see Fig. 1(ii). Here the prefix of length $e_y - c_x$ of x is a suffix of y and has period $q - p$.

(iii) $i_y < i_x < c_x < c_y (< e_x < e_y)$. Here x and the prefix of length $c_x - i_y$ of y have period $q - p$; see Fig. 1(iii). As above, a third δ-close run z^2 would have to share the same beginning with y^2, otherwise one of y^2 or z^2 would be left extendable. A fourth δ-close run would have to start at the same place and, because of the three-prefix-square lemma[2] of [3], since p is primitive, it would have a period at least $q + r$, which is impossible.

(iv) $i_x < i_y (< c_x < c_y < e_x < e_y)$; this is similar with (iii); see Fig. 1(iv). A third run would begin at the same position as y^2 and there is no fourth run.

(v) $i_x = i_y$; see Fig. 1(v). Here not even a third δ-close run exists because of the three-square lemma that implies $r \geq p + q$.

(vi) $c_x = c_y$. This case is significantly different from the other ones, as we can have many δ-close runs here. However, the existence of many runs with the same center implies very strong periodicity properties of the string which allow us to count the runs globally and obtain even fewer runs than before.

In this case both x and y have the same small period $\ell = q - p$; see Fig. 1(vi). If we note $c = c_y$ then we have h runs $x_j^{\alpha_j}$, $1 \leq j \leq h$, beginning at positions $i_{x_j} = c - ((j-1)\ell + \ell')$, where ℓ' is the length of the suffix of x that is a prefix of the period.

We show that in this case we have less runs than as counted in the sum (3.1). For $h \leq 9$ there is nothing to prove as no four of our $x_j^{\alpha_j}$ runs are counted for the same δ. Assume $h \geq 10$. There exists δ_i such that $\frac{\ell}{2} \leq \delta_i \leq \frac{3\ell}{4}$, that is, this δ_i is considered in (3.1). Then

[2]For three words u, v, w, it states that if uu is a prefix of vv, vv is a prefix of ww, and u is primitive, then $|u| + |v| \leq |w|$.

it is not difficult to see that there is no run in w with period between ℓ and $\frac{9}{4}\ell$ and center inside $J = [c + \ell + 1 \,..\, c + (h-2)\ell + \ell']$. But $\ell \le 2\delta_i < 3\delta_i \le \frac{9}{4}\ell$ and the length of J is $(h-3)\ell + \ell' \ge (h+1)\delta_i$. This means that at least h intervals of length δ_i in the sum (3.1) are covered by J and therefore at least $3h$ runs in (3.1) are replaced by our h runs.

We need also mention that these h intervals of length δ_i are not reused by a different center with multiple runs since such centers cannot be close to each other. Indeed, if we have two centers c_j with the above parameters h_j, ℓ_j, $j = 1, 2$, then, as soon as the longest runs overlap over $\ell_1 + \ell_2$ positions, we have $\ell_1 = \ell_2$, due to Fine and Wilf's lemma. Then, the closest positions of J_1 and J_2 cannot be closer than $\ell_1 = \ell_2 \ge \delta_i$ as this would make some of the runs non-primitive, a contradiction. Thus the bound in (3.1) still holds and we proved

Theorem 3.1. *The number of runs in a string of length n is $\mathcal{O}(n)$.*

4. The sum of exponents

Using the above approach, we show in this section that the sum of exponents of all runs is also linear. The idea is to prove that the sum of exponents of all runs with the centers in an interval of length δ and periods between 2δ and 3δ is less than 8. (As in the previous proof, there are exceptions to this rule, but in those cases we get a smaller sum of exponents.) Then a computation similar to (3.1) gives that the sum of exponents is at most $48n$.

To start with, Fine and Wilf's periodicity lemma can be rephrased as follows: For two primitive strings x and y, any powers x^α and y^β cannot have a common factor longer than $|x| + |y|$ as such a factor would have also period $\gcd(|x|, |y|)$, contradicting the primitivity of x and y.

Next consider two δ-close runs, x^α and y^β, $\alpha, \beta \in \mathbb{Q}$. It cannot be that both α and β are 2.5 or larger, as this would imply an overlap of length at least $|x| + |y|$ between the two runs, which is forbidden by Fine and Wilf's lemma since x and y are primitive. Therefore, in case we have three mutually δ-close runs, two of them must have their exponents smaller than 2.5. If the exponent of the third run is less than 3, we obtain the total of 8 we were looking for. However, the third run, say z^γ, $\gamma \in \mathbb{Q}$, may have a larger exponent. If it does, that affects the runs in the neighboring intervals of length δ. More precisely, if $\gamma \ge 3$, then there cannot be any center of run with period between 2δ and 3δ in the next (to the right) interval of length δ. Indeed, the overlap between any such run and z^γ would imply, as above, that their roots are not primitive, a contradiction. In general, the following $\lfloor 2(\gamma - 2.5) \rfloor$ intervals of length δ cannot contain any center of such runs. Thus, we obtain a smaller sum of exponents when this situation is met.

The second exception is given by case (vi) in the previous proof, that is, when many runs share the same center; we use the same notation as in (vi). We need to be aware of the exponent of the run $x_1^{\alpha_1}$, with the smallest period, as α_1 can be as large as ℓ (and unrelated to h, the number of runs with the same center). We shall count α_1 into the appropriate interval of length δ_i; notice that $x_1^{\alpha_1}$ and $x_2^{\alpha_2}$ are never δ-close, for any δ, because $|x_2| > 2|x_1|$. For $2 \le j \le h-1$, the period $|x_j|$ cannot be extended by more than ℓ positions to the right past the end of the initial square, and thus $\alpha_j \le 2 + \frac{1}{j}$. Therefore, their contribution to the sum of exponents is less than $3(h-2)$. They replace the exponents of the runs with centers in the interval J and periods between ℓ and $\frac{9}{4}\ell$ which otherwise would contribute at least

$6h$ to the sum of exponents. The run with the longest period, $x_h^{\alpha_h}$, can have an arbitrarily high exponent but the replaced runs in J need to account only for a fraction (3 units) of it since $\alpha_h \geq 3$ implies new centers with multiple runs and hence new J intervals (precisely $\lfloor \alpha_h - 2 \rfloor$) that account for the rest. We proved

Theorem 4.1. *The sum of exponents of the runs in a string of length n is $\mathcal{O}(n)$.*

References

[1] M. Crochemore, An optimal algorithm for computing the repetitions in a string, *Inform. Proc. Letters* **12** (1981) 244 – 250.
[2] M. Crochemore and L. Ilie. Maximal repetitions in strings. *Journal of Computer and System Sciences*, 2007. In press.
[3] M. Crochemore and W. Rytter, Squares, cubes, and time-space efficient string searching, *Algorithmica* **13** (1995) 405 – 425.
[4] C.S. Iliopoulos, D. Moore, W.F. Smyth, A characterization of the squares in a Fibonacci string, *Theoret. Comput. Sci.* **172** (1997) 281 – 291.
[5] R. Kolpakov and G. Kucherov, On the sum of exponents of maximal repetitions in a word, Tech. Report 99-R-034, LORIA, 1999.
[6] R. Kolpakov and G. Kucherov, Finding maximal repetitions in a word in linear time, *Proc. of FOCS'99*, IEEE Computer Society Press, 1999, 596 – 604.
[7] R. Kolpakov and G. Kucherov, On maximal repetitions in words, *J. Discrete Algorithms* **1**(1) (2000) 159 – 186.
[8] M. Lothaire, *Algebraic Combinatorics on Words*, Cambridge Univ. Press, 2002.
[9] M.G. Main, Detecting lefmost maximal periodicities, *Discrete Applied Math.* **25** (1989) 145 – 153.
[10] W. Rytter, The number of runs in a string: improved analysis of the linear upper bound, in: B. Durand and W. Thomas (eds.), *Proc. of STACS'06*, Lecture Notes in Comput. Sci. **3884**, Springer-Verlag, Berlin, 2006, 184 – 195.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 17-18
www.stacs-conf.org

A LITTLE BIT INFINITE? ON ADDING DATA TO FINITELY LABELLED STRUCTURES (ABSTRACT)

THOMAS SCHWENTICK

Universitt Dortmund, Lehrstuhl Informatik I, 44221 Dortmund, Germany
E-mail address: thomas.schwentick@udo.edu

Finite or infinite strings or trees with labels from a finite alphabet play an important role in computer science. They can be used to model many interesting objects including system runs in Automated Verification and XML documents in Database Theory. They allow the application of formal tools like logical formulas to specify properties and automata for their implementation. In this framework, many reasoning tasks that are undecidable for general computational models can be solved algorithmically, sometimes even efficiently.

Nevertheless, the use of finitely labelled structures usually requires an early abstraction from the *real* data. For example, theoretical research on XML processing very often concentrates on the document structure (including labels) but ignores attribute or text values. While this abstraction has led to many interesting results, some aspects like key or other integrity constraints can not be adequately handled.

In Automated Verification of software systems or communication protocols, infinite domains occur even more naturally, e.g., induced by program data, recursion, time, communication or by unbounded numbers of concurrent processes. Usually one approximates infinite domains by finite ones in a very early abstraction step.

An alternative approach that has been investigated in recent years is to extend strings and trees by (a limited amount of) data and to use logical languages with a restricted expressive power concerning this data. As an example, in the most simple setting, formulas can only test equality of data values. The driving goal is to identify logical languages and corresponding automata models which are strong enough to describe interesting properties of data-enhanced structures while keeping decidability or even feasibility of automatic reasoning.

The talk gives a basic introduction into data-enhanced finitely labelled structures, presents examples of their use, and highlights recent decidability and complexity results.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 19-38
www.stacs-conf.org

EQUILIBRIA, FIXED POINTS, AND COMPLEXITY CLASSES

MIHALIS YANNAKAKIS

Department of Computer Science, Columbia University, New York City, NY, USA
E-mail address: mihalis@cs.columbia.edu

Abstract. Many models from a variety of areas involve the computation of an equilibrium or fixed point of some kind. Examples include Nash equilibria in games; market equilibria; computing optimal strategies and the values of competitive games (stochastic and other games); stable configurations of neural networks; analysing basic stochastic models for evolution like branching processes and for language like stochastic context-free grammars; and models that incorporate the basic primitives of probability and recursion like recursive Markov chains. It is not known whether these problems can be solved in polynomial time. There are certain common computational principles underlying different types of equilibria, which are captured by the complexity classes PLS, PPAD, and FIXP. Representative complete problems for these classes are respectively, pure Nash equilibria in games where they are guaranteed to exist, (mixed) Nash equilibria in 2-player normal form games, and (mixed) Nash equilibria in normal form games with 3 (or more) players. This paper reviews the underlying computational principles and the corresponding classes.

1. Introduction

Many situations involve the computation of an equilibrium or a stable configuration of some sort in a dynamic environment. Sometimes it is the result of individual agents acting on their own noncompetitively but selfishly (e.g., Nash and other economic equilibria), sometimes it is agents acting competitively against each other (and perhaps nature/chance), sometimes the equilibrium is the limit of an iterative process that evolves in some direction until it settles. Often the sought objects can be described mathematically as the fixed points of an equation $x = F(x)$.

Many models and problems from a broad variety of areas are of this nature. Examples include: Nash equilibria in games; market equilibria; computation of optimal strategies and the values of competitive games (stochastic and other games); stable configurations of neural networks; analysis of basic stochastic models for evolution like branching processes, and for language like stochastic context-free grammars; and models that incorporate the basic primitives of probability and recursion like recursive Markov chains. Most of these models and problems have been studied mathematically for a long time, leading to the

1998 ACM Subject Classification: F.1.3, F.2.
Key words and phrases: Equilibria, Fixed points, Computational Complexity, Game Theory.
Work supported by NSF Grant CCF-0728736.

development of rich theories. Yet, some of their most basic algorithmic questions are still not resolved, in particular it is not known whether they can be solved in polynomial time.

Despite the broad diversity of these problems, there are certain common computational principles that underlie many of these different types of problems, which are captured by the complexity classes PLS, PPAD, and FIXP. In this paper we will review these principles, the corresponding classes, and the types of problems they contain.

All the problems we will discuss are total search problems. Formally, a *search problem* Π has a set of instances, each instance I has a set $Ans(I)$ of acceptable answers; the search problem is *total* if $Ans(I) \neq \emptyset$ for all instances I. As usual, for computational purposes, instances are represented by strings over a fixed alphabet Σ, and it is assumed that, given a string over Σ one can determine in polynomial time if it represents an instance of a problem. The size $|I|$ of an instance is the length of its string representation. Input numbers (such as the payoffs of games, input probabilities of stochastic models, etc.) are assumed to be rationals represented in binary by numerator and denominator. The underlying solution space from which answers are drawn may be finite and discrete, as in combinatorial problems, or it may be infinite and continuous. In the former (the finite) case, solutions are represented also as strings and the problem is: given an instance I, compute a solution in $Ans(I)$. In the latter (infinite/continuous) case also, if there are rational-valued solutions (as in Linear Programming for example), then the problem is to compute one of them. In several problems however, the solutions are inherently irrational, and we cannot compute them exactly (in the usual Turing machine-based model of computation and complexity). In these cases we need to specify precisely which information about the solutions is to be computed; this could be for example a yes/no question, such as, does an event in a stochastic model occur almost surely (with probability 1) or does the value of a game exceed a given threshold, or we may want to compute an answer up to a desired precision. In any case, the computational tasks of interest have to be defined precisely, because different tasks can have different complexity.

In this paper we will discuss a variety of equilibria and fixed point problems, and the complexity classes which capture the essential aspects of several types of such problems. We discuss three classes, PLS, PPAD, and FIXP, which capture different types of equilibria. Some representative complete problems for these classes are: for PLS pure Nash equilibria in games where they are guaranteed to exist, for PPAD (mixed) Nash equilibria in 2-player normal form games, and for FIXP (mixed) Nash equilibria in normal form games with 3 (or more) players.

2. Discrete, Pure Equilibria and the Class PLS

Consider the following *neural network* model [34]: We have an undirected graph $G = (V, E)$ with a positive or negative weight $w(e)$ on each edge $e \in E$ (we can consider missing edges as having weight 0) and a threshold $t(v)$ for each node $v \in V$. A configuration of the network is an assignment of a state $s(v) = +1$ ('on') or -1 ('off') to each node $v \in V$. A node v is *stable* (or 'happy') if $s(v) = 1$ and $\sum_u w(v,u)s(u) + t(v) \geq 0$, or $s(v) = -1$ and $\sum_u w(v,u)s(u) + t(v) \leq 0$, i.e. the state of v agrees with the sign of the weighted sum of its neighbors plus the threshold. A configuration is *stable* if all the nodes are stable. A priori it is not obvious that such a configuration exists; in fact for directed networks there may not exist any. However, every undirected network has at least one (or more) stable configuration [34]. In fact, a dynamic process where in each step one node that is unstable (any one)

switches its state, is guaranteed to eventually converge in a finite number of steps to a stable configuration, no matter which unstable node is switched in each step. (It is important that updates be asynchronous, one node at a time; simultaneous updates can lead to oscillations.) To show the existence of a stable configuration and convergence, Hopfield introduced a value function (or 'potential' or 'energy') on configurations, $p(s) = \sum_{(v,u)\in E} w(v,u)s(v)s(u) + \sum_{v\in V} t(v)s(v)$. If v is an unstable node in configuration s, then switching its state results in a configuration s' with strictly higher value $p(s') = p(s)+2|\sum_u w(v,u)s(u)+t(v)| > p(s)$. Since there is a finite number ($2^{|V|}$) of configurations, the process has to converge to a stable configuration. The *stable configuration problem* is the following: Given a neural network, compute a stable configuration. This is a total search problem, as there may be one or more stable configurations, and anyone of them is an acceptable output.

Although the stable configuration problem does not call a priori for any optimization, the problem can be viewed equivalently as one of *local* optimization: compute a configuration s whose value $p(s)$ cannot be increased by switching the state of any single node. Local search is a common, general approach for tackling hard optimization problems. In a combinatorial optimization problem Π, every instance I has an associated finite set $S(I)$ of solutions, every solution $s \in S(I)$ has a rational value or cost $p_I(s)$ that is to be maximized or minimized. In local search, a solution $s \in S(I)$ has in addition an associated neighborhood $N_I(s) \subseteq S(I)$; a solution is locally optimal if it does not have any (strictly) better neighbor, i.e. one with higher value or lower cost. A standard local search algorithm starts from an initial solution, and keeps moving to a better neighbor as long as there is one, until it reaches a local optimum. The complexity class PLS (Polynomial Local Search) was introduced in [36] to capture the inherent complexity of local optima for usual combinatorial problems, where each step of the local search algorithm can be done in polynomial time. Even though each step takes polynomial time, the number of steps can be potentially exponential, and in fact for many problems we do not know how to compute even locally optimal solutions in polynomial time. Formally, a problem Π is in PLS if solutions are polynomially bounded in the input size, and there are polynomial-time algorithms for the following tasks: (a) test whether a given string I is an instance of Π and if so compute a (initial) solution in $S(I)$, (b) given I, s, test whether $s \in S(I)$ and if so compute its value $p_I(s)$, (c) given I, s, test whether s is a local optimum and if not, compute a better neighbor $s' \in N_I(s)$. Notions of PLS reduction and completeness were introduced to relate the problems. A number of well-studied combinatorial optimization problems (e.g. Graph Partitioning, TSP, Max Cut, Max Sat etc.) with common neighborhood structures (both simple and sophisticated) have been shown to be PLS-complete by many researchers, and thus locally optimal solutions can be computed efficiently for anyone of them iff they can be computed for all PLS problems. For a detailed survey and bibliography see [66]. In particular, the stable configuration problem is PLS-complete (and is complete even if all thresholds are 0 and all weights are negative, i.e. all connections are repulsive) [58].

It is worth stressing several points: 1. The search problem asks to compute any local optimum, not a specific one like the best, which is often NP-hard. 2. Given an instance I, we can always guess a solution s, and verify in polynomial time that it is indeed a solution ($s \in S(I)$) and it is locally optimal. Hence PLS is somewhere between P and TFNP (total search problems in NP). Such problems cannot be NP-hard (under Cook reductions) unless NP=coNP. 3. We are interested in the inherent complexity of the search probleme itself *by any algorithm whatsoever*, not necessarily the standard local search algorithm, which often has exponential running time. For example, Linear Programming can be viewed as

a local search problem (where local optima= global optima) with Simplex as the local search algorithm; we know that Simplex under many pivoting rules is exponential, yet the problem itself can be solved in polynomial time by completely different methods (Ellipsoid, Karmakar). In fact, many common local search problems are complete under a type of *tight* PLS-reduction which allows us to conclude that the corresponding standard local search algorithm is exponential. For example, in the neural network model, the dynamic process where unstable nodes switch iteratively their state until the network stabilizes takes for some networks and for some (in fact for most) initial configurations exponential time to converge, no matter which unstable node is switched in each step. Furthermore, the computational problem: given a network and initial configuration compute a stable configuration (anyone) that can result from this process is a PSPACE-complete problem.

Another type of equilibrium problems that can be placed in PLS concerns finding pure Nash equlibria for games where they are guaranteed to exist. A (finite) game has a finite set k of players, each player $i = 1, \ldots, k$, has a finite set S_i of pure strategies and a payoff (utility) function U_i on the product strategy space $S = \Pi_i S_i$; we assume for computational purposes that U_i takes rational values. A pure strategy profile s is a member of S, i.e. a choice of a pure strategy $s_i \in S_i$ for each player. It is a pure Nash equilibrium if no player can improve his payoff by switching unilaterally to another pure strategy; that is, if (s_{-i}, s'_i) denotes the profile where player i plays strategy $s'_i \in S_i$ and the other players play the same strategy as in s, then $U_i(s) \geq U_i(s_{-i}, s'_i)$ for every i and every $s'_i \in S_i$. Not every game has a pure Nash equilibrium. A *mixed strategy* for player i is a probability distribution on S_i. Letting M_i denote the set of mixed strategies for player i, the set of mixed strategy profiles is their product $M = \Pi_i M_i$; i.e., a mixed strategy profile is a non-negative vector x of length $\sum_i |S_i|$ (i.e. its entries are indexed by all the players' pure strategies) that is a probability distribution on the set of pure strategies of each player. The (expected) payoff $U_i(x)$ of x for player i is $\sum x_{1,j_1} \ldots x_{k,j_k} U_i(j_1, \ldots, j_k)$ where the sum is over all tuples $(j_1, \ldots, j_k)$ such that $j_1 \in S_1, \ldots, j_k \in S_k$, and $x_{i,j}$ is the entry of x defining the probability with which player i plays strategy j. A (mixed) *Nash equilibrium* (NE) is a strategy profile x^* such that no player can increase its payoff by switching to another strategy unilaterally. Every finite game has at least one Nash equilibrium [46].

For example, a neural network can be viewed as a game with one player for each node, each player has two pure strategies $+1, -1$ (corresponding to the two states) and its payoff function has two values 1 (happy) and 0 (unhappy) depending on its state and that of its adjacent nodes. The stable configurations of the network are exactly the pure Nash equilibria of the game. This game is a case of a *graphical game*: players correspond to nodes of a graph and the payoff function of a player depends only on its own strategy and that of its neighbors. General graphical games may not have pure Nash equilibria. For an overview of graphical games see [39].

There is a class of games, *congestion games*, in which there is always a pure equilibrium. In a congestion game, there are k players, a finite set R of resources, the pure strategy set $S_i \subseteq 2^R$ of each player is a family of subsets of the resources, each resource $r \in R$ has an associated cost function $d_r : \{0, \ldots, k\} \to Z$. If $s = (s_1, \ldots, s_k)$ is a pure strategy profile, the congestion $n_r(s)$ of a resource r is the number of players whose strategy contains r; the cost (negative payoff) of a player i is $\sum_{r \in s_i} d_r(n_r(s))$. Rosenthal showed that every congestion game has a pure equilibrium [54]. In fact, the iterative process where in each step, if the current pure strategy profile is not at equilibrium, a player with a suboptimal strategy switches to a strategy with a lower cost (while other players keep the same strategy)

does not repeat any profile and thus converges in a finite number of steps to an equilibrium. The proof is by introducing a potential function $p(s) = \sum_{r \in R} \sum_{i=1}^{n_r(s)} d_r(i)$ and showing that switching the strategy of a player to a lower cost strategy results in a reduction of the potential function by the same amount. Thus, the pure equilibria are exactly the local optima of the potential function $p(s)$ with respect to the neighborhood that switches the strategy of a single player. Computing a pure equilibrium is a local search problem, and it is in PLS provided that the costs functions d_r of the resources are polynomial time computable, and the strategy sets S_i are given explicitly or at least one can determine efficiently whether a player can improve his strategy for a given profile. Furthermore, Fabrikant et al. [27] showed that the problem is PLS-complete. They showed that it is complete even in the case of *network congestion games*, where the resources are the edges of a given directed graph, each player i has an associated source s_i and target node t_i and its set S_i of pure strategies is the set of $s_i - t_i$ paths; the cost function d_r of each edge r represents the delay as a function of the paths that use the edge, and completeness holds even for linear delay functions [1]. As with other PLS-complete problems, a consequence of the reductions, which are tight, is that the iterative local improvement algorithm can take exponential time to converge. For more information on congestion games see [64].

There are several other games which are in PLS and not known to be in P, and which are not known (and not believed to be) PLS-complete. These are not one-shot games, but they are dynamic games played iteratively over time (like chess, backgammon etc.). There are two main types of payoffs for the players in such games: in one type, the payoff of a history is an aggregation of rewards obtained in the individual steps of the history combined via some aggregation function, such as average reward per step or a discounted sum of the rewards; in the other type, the payoff obtained depends on the properties of the history. We will discuss three such games in this section, and some more in the following sections.

A *simple stochastic game* [13] is a 2-player game played on a directed graph $G = (V, E)$ whose nodes represent the positions of the game, and the edges represent the possible moves. The sinks are labelled 1 or 2 and the nonsink nodes are partitioned into three sets, V_r (random nodes), V_1 (max or player 1 nodes), V_2 (min or player 2 nodes); the edges (u, v) out of each random node u are labelled with probabilities p_{uv} (assumed to be rational for computational puposes) that sum to 1. Play starts at some initial node (position) and then moves in each step along the edges of the graph; at a random node the edge is chosen randomly, at a node of V_1 it is chosen by player 1, and at a node of V_2 it is chosen by player 2. If the play reaches a sink labelled 1, then player 1 is the winner, while if it reaches a sink labelled 2 or it goes on forever, then player 2 is the winner. The goal of player 1 is to maximize her probability of winning, and the goal of player 2 is to minimize it (i.e. maximize his own winning probability). These are zero-sum games (what one player wins the other loses). For every starting node s there is a well-defined value x_s of the game, which is the probability that player 1 wins if they both play optimally. Although the players are allowed to use randomization in each step and have their choice depend on their entire history, it is known that there are stationary, pure (deterministic) optimal strategies for both players. Such a strategy σ_i for player $i = 1, 2$ is simply a choice of an outgoing edge (a successor) for each node in V_i, thus there is a finite number of such pure strategies. For every pure strategy profile (σ_1, σ_2) for the two players, the game reduces to a Markov chain and the values $x_s(\sigma_1, \sigma_2)$ can be computed by solving a linear system of equations. If the edge probabilities are rational then the optimal values x_s are also rational, of bit complexity polynomial in the input size. If there are only two of the three types of nodes in the graph,

then the optimal strategies and the values x_s can be computed in polynomial time. For example, if there is no player 2, then the game becomes a Markov decision process with the goal of maximizing the probability of reaching a sink labelled 1, which can be optimized by Linear Programming. When we have all three types of nodes, the decision problem $x_s \geq 1/2$? (does player 1 win with probability at least 1/2 starting from position s) is in $NP \cap coNP$ (in fact in $UP \cap coUP$), and it is a well-known open problem whether it is in P [13]. Two (pure) strategies σ_1, σ_2 of the two players form an *equilibrium* if σ_1 is a best response of player 1 to the strategy σ_2 of player 2 (i.e. σ_1 is a maximizing strategy in the Markov decision process obtained when the strategy of player 2 is fixed to σ_2), and vice-versa, σ_2 is a best response of player 2 to σ_1. The equilibria are precisely the optimal strategy pairs. The problem can be viewed as a local search problem in PLS if we take the point of view of one player, say player 1: the solution set is the set of pure strategies of player 1, the value of a strategy σ_1 is $\sum_{s \in V} x_s(\sigma_1, \sigma_2)$ where σ_2 is a best response of player 2 to σ_1, and the neighbors of σ_1 are the strategies obtained by switching the choice of a node in V_1. The locally optimal solutions are the (globally) optimal strategies of player 1.

A *mean payoff* game [18] is a non-stochastic 2-player game played on a directed graph $G = (V, E)$ with no sinks, whose nodes are partitioned into two sets V_1, V_2 and whose edges are labelled by (rational) rewards $r(e), e \in E$. As above, play starts at a node and moves along the edges, where player 1 chooses the next edge for nodes in V_1 and player 2 for nodes in V_2 (there are no random nodes here), and play goes on forever. The payoff to player 1 from player 2 of a history using the sequence of edges $e_1, e_2, \ldots$ is the average reward per step, $\limsup_{n\to\infty}(\sum_{j=1}^{n} r(e_j))/n$. Again there are optimal pure stationary strategies σ_1, σ_2 for the players, and these form a path followed by a cycle C; the payoff (value of the game) is the ratio $\sum_{e \in C} r(e)/|C|$ and is rational of polynomial bit complexity. As shown in [67], the optimal values and optimal strategies can be computed in pseudopolynomial time (i.e. polynomial time for unary rewards); furthermore the problem can be reduced to simple stochastic games, it is thus in PLS and the decision problem is in $UP \cap coUP$, but it is open whether it is in P.

A still simpler, nonstochastic 2-player game, called *parity game* [20] has been studied extensively in the verification area; it is an important theoretical question in this area whether this game can be solved in polynomial time. A parity game is played again on a directed graph G whose nodes are partitioned into two sets V_1, V_2 and whose edges are labelled by positive integers. A history is winning for player 1 (respectively player 2) if the maximum label that occurs infinitely often in the history is odd (resp. even). In this game, one of the two players has a pure optimal strategy that wins on every history that results against every strategy of the other player. Determining who the winner of the game is (and a winning strategy) reduces to the decision problem for mean payoff games and in turn to simple stochastic games [51, 38].

3. Fixed Points

Nash's theorem asserts that every finite game Γ has a (generally, mixed) equilibrium. Nash proved his theorem in [46] using Brouwer's fixed point theorem: every continuous function F from a compact convex body to itself has a fixed point, i.e. a point x such that $x = F(x)$. Specifically, given a finite game Γ with k players $i = 1, \ldots, k$, a finite set S_i of pure strategies and a payoff function U_i for each player, a mixed strategy profile is a vector $x = (x_{ij} | i = 1, \ldots, k; j = 1, \ldots, |S_i|)$, which lies on the product Δ of the k unit simplexes

$\Delta_i = \{y \in R^{|S_i|} | \sum_{j=1}^{|S_i|} y_j = 1; y \geq 0\}$. Nash defined the following function from Δ to itself: $F_\Gamma(x)_{(i,j)} \doteq \frac{x_{i,j}+\max\{0,g_{i,j}(x)\}}{1+\sum_{l=1}^{|S_i|}\max\{0,g_{i,l}(x)\}}$, where $g_{i,j}(x)$ is the (positive or negative) "gain" in payoff of player i if he switches to pure strategy j while the other players continue to play according to x; $g_{i,j}(x)$ is a (multivariate) polynomial in x. Nash showed that the fixed points of F_Γ are precisely the equilibria of the game Γ. There are several alternative proofs of Nash's theorem, all using Brouwer's theorem (with different functions F) or the related Kakutani's theorem (for fixed points of multivalued maps). Note that the underlying solution space here, Δ, is continuous, not discrete and finite. Furthermore, even if the payoff functions of the game are rational-valued, for 3 or more players it may be the case that all equilibria are irrational.

Market equilibria is another important application of fixed point theorems. Consider the following exchange model [57]. We have m agents and n commodities. The agents come to the market with an initial supply of commodities, which they exchange for their prefered ones; each agent sells his supply at the prevailing prices, and buys his preferred bundle of commodities. For each vector p of prices for the commodities, each agent ℓ has an (positive or negative) 'excess demand' (=demand-supply) $g_i^\ell(p)$ for each commodity i. Standard assumptions are that the functions $g_i^\ell(p)$ (i) are homogeneous of degree 0, thus the price vectors may be normalized to lie on the unit simplex Δ_n, (ii) they satisfy Walras' law $\sum_{i=1}^n p_i g_i^\ell(p) = 0$, (iii) they are continuous on the unit simplex. Let $g_i(p) = \sum_\ell g_i^\ell(p)$ be the (total) market excess demand for each commodity i. The functions $g_i(p)$ satisfy the same constraints. A vector p of prices is an *equilibrium* if $g_i(p) \leq 0$ for all i (demand does not exceed supply), with equality for all commodities i that have $p_i > 0$. Brouwer's theorem can be used to show the existence of equilibria. Namely, the equilibria are the fixed points of the function $F : \Delta_n \mapsto \Delta_n$, defined by the formula $F_i(p) = \frac{p_i+\max(0,g_i(p))}{1+\sum_{j=1}^n \max(0,g_j(p))}$. In fact, the equilibrium existence theorem can be conversely used to show Brouwer's theorem: from a Brouwer function one can construct an economy whose equilibria correspond to the fixed points of the function [63]. In the classical Arrow-Debreu market model [3], the user preferences for the commodities are modeled by utility functions, which in turn induce the excess demand functions (or correspondences, i.e. multivalued maps), and more generally the model includes also production. Under suitable conditions, the existence of equilibria is derived again using a fixed point theorem (Kakutani in [3], or Brouwer in alternative proofs [29]). As shown in a line of work by Sonnenschein, Mantel, Debreu and others (see e.g. [17]), essentially any function satisfying the standard conditions can arise as the excess demand function in a market for suitably defined utility functions for the users . Thus, there is a tight connection between fixed points of general functions and market equilibria.

A number of other problems from various domains can be cast as fixed point computation problems, i.e., every instance I of a problem is associated with a function F_I over some domain so that the sought objects $Ans(I)$ are fixed points of F_I; in some cases, we may only want a specific fixed point of the function. We will mention several more examples in this section. Recall the simple stochastic game from the last section. The vector $x = (x_s | s \in V)$ of winning probabilities for Player 1 satisfies the following system of equations $x = F(x)$, with one equation for each node s: if s is a sink labelled 1 (respectively 2) then $x_s = 1$ (resp. $x_s = 0$); if $s \in V_r$ then $x_s = \sum_{(s,v)\in E} p_{sv} x_v$; if $s \in V_1$ then $x_s = \max\{x_v | (s,v) \in E\}$; if $s \in V_2$ then $x_s = \min\{x_v | (s,v) \in E\}$. In general there may be multiple solutions, however the system can be preprocessed so that there is a unique solution in the unit cube $C_n = \{x | 0 \leq x_s \leq 1, \forall s \in V\}$.

Stochastic games were originally introduced by Shapley in [59] in a more general form, where players can move simultaneously. As shown in [13], simple stochastic games can be reduced to Shapley's game. In *Shapley's game* there is a finite set V of states, each state u has an associated one-shot zero-sum finite game with a reward (payoff) matrix A_u whose rows (resp. columns) correspond to the actions (pure strategies) of Player 1 (resp. 2). If the play is in state u and the players choose actions i, j then Player 1 receives reward $A_u[i, j]$ from Player 2, the game stops with probability $q^u_{ij} > 0$, and it transitions to state v with probability p^{uv}_{ij}, where $q^u_{ij} + \sum_v p^{uv}_{ij} = 1$. Since there is at least positive probability $q = \min\{q^u_{ij} | u, i, j\} > 0$ of stopping in each step, the game stops a.s. in a finite number of steps. (Another standard equivalent formulation is as a discounted game, where the game does not stop but future rewards are discounted by a factor $1 - q$ per step). The goal of Player 1 is to maximize (and of Player 2 to minimize) the total expected reward, which is the value of the game. We want to compute the vector $x = (x_u | u \in V)$ of game values for the different starting states u. As usual all rewards and probabilities are assumed to be rationals for computational purposes. The values in general may be irrational now however. The vector x satisfies a fixed point set of equations $x = F(x)$, as follows. For each state u, let $B_u(x)$ be the matrix, indexed by the actions of the players, whose i, j entry is $A_u[i, j] + \sum_v p^{uv}_{ij} x_v$, and let $Val(B_u(x))$ be the value of the one-shot zero-sum game with payoff matrix $B_u(x)$. Then $x = F(x)$ where $F_u(x) = Val(B_u(x)),\ u \in V$. The function F is a Banach function (a contraction map) under the L_∞ norm with contraction factor $1-q$, and thus it has a unique fixed point, the vector of values of the game.

Branching processes are a basic model of stochastic evolution, introduced first in the single type case by Galton and Watson in the 19th century to study population dynamics, and extended later to the multitype case by Kolmogorov and Sevastyanov, motivated by biology. A branching process has a finite set T of n types, for each type $i \in T$ there is a finite set of 'reproduction' rules of the form $i \stackrel{p_{ij}}{\to} v_{ij}, j = 1, \ldots, m_i$, where $p_{ij} \in [0, 1]$ is the probability of the rule (thus, $\sum_{j=1}^{m_i} p_{ij} = 1$) and $v_{ij} \in \mathbb{N}^n$ is a vector whose components specify the number of offsprings of each type that an entity of type i produces in the next generation. Starting from an initial population, the process evolves from one generation to the next according to the probabilistic reporuction rules. The basic quantity of interest is the probability x_i of extinction of each type: the probability that if we start with one individual of type i, the process will eventually die. These can be used to compute the extinction probability for any initial population and are the basic for more detailed statistics of the process. As usual, we assume that the probabilities of the rules are rational. However the extinction probabilities are in general irrational. The vector x satisfies a set of fixed point equations $x = F(x)$, where $F_i(x)$ is the polynomial $\sum_{j=1}^{m_i} p_{ij} \Pi_{k=1}^{n} (x_k)^{v_{ij}[k]}$. Note that $F_i(x)$ has positive coefficients, thus F is a monotone operator on $\mathbb{R}^n_{\geq 0}$ and thus has a Least Fixed Point (LFP); the LFP is precisely the vector of extinction probabilities of the branching process. For more information on the theory of branching processes and their applications see [32, 31].

Stochastic context-free grammars (SCFG) are context-free grammars where the production rules have associated probabilities. They have been studied extensively in Natural Language Processing where they are an important model [45], and have been used also in biological sequence analysis. A basic quantity of interest is the probability of the language generated by a SCFG; again this may be an irrational number even if all the probabilities

of the production rules are rational. The analysis of SCFG's is closely related to that of branching processes.

A model that encompasses and generalizes both of branching processes and SCFG's in a certain precise sense, is the *Recursive Markov chains* (RMC) model [22] and the equivalent model of *Probabilistic Pushdown machines* [21]. Informally, a RMC is a collection of Markov chains that can call each other in a potentially recursive manner like recursive procedures. The basic quantities of interest are the termination probabilities. These probabilities obey again a system of fixed point equations $x = F(x)$, where F is a vector of polynomials with positive coefficients; the least fixed point of the system gives the termination probabilities of the RMC. Generalization to a setting where the dynamics are not completely probabilistic but can be controlled by one or more players leads to *recursive Markov decision processes and games* [24, 25, 23]. For example, we may have a branching process, where the reproduction can be influenced by players who want to bias the process towards extinction or survival. This results in fixed point systems of equations involving monotone polynomials and the min and max operators.

All of the above problems are total (single-valued or multi-valued) search problems, in which the underlying solution space is continuous. In all of these problems we would ideally like to compute exactly the quantities of interest if possible (if they are rational), and otherwise, we would like to bound them and answer decision questions about them (eg. is the value of a stochastic game $\geq 1/2$?, does a RMC terminate with probability 1?) or to approximate them within desired precision, i.e. compute a solution x that is within ϵ of an/the answer x^* to the search problem (eg., approximate within additive error ϵ the extinction probabilities of a branching process, or compute a mixed strategy profile for a game that is within ϵ of a Nash equilibrium). In the approximation problem we would like ideally polynomial time in the size of the input and in $\log(1/\epsilon)$ (the number of bits of precision). We refer to the approximation of an answer to a search problem as above as *strong* approximation (or the 'near' problem) to distinguish it from another notion of approximation, which we call the *weak* approximation (or the 'almost' problem) that is specific to a fixed point formulation of a search problem via a function F: a weak ϵ-approximation is a point x such that $|x - F(x)| \leq \epsilon$ (say in the L_∞ norm). Note that a search problem may be expressible in different ways as a fixed point problem using different functions F, and the notion of weak approximation may depend on the function that is used; the strong approximation notion is intrinsic to the search problem itself (does not depend on F). For many common fixed point problems (formally, for polynomially continuous functions [26]), including all of the above problems, weak approximation reduces to strong, i.e., given instance I and (rational) $\epsilon > 0$, we can define a (rational) $\delta > 0$ of bit-size polynomial in that of ϵ and in $|I|$ such that every (strong) δ-approximation x to an answer to the search problem (i.e., approximation to a fixed point of the function F_I corresponding to the instance I) is a weak ϵ-approximate fixed point (i.e., satisfies $|x - F_I(x)| \leq \epsilon$). The converse relation does not hold in general; in particular, it does not hold for Nash equilibria and the Nash function F_Γ.

We discuss briefly now algorithms for such fixed point problems. For a Banach function F_I we can start at any point x_0, and apply repeatedly F_I. The process will converge to the unique fixed point. If the contraction factor $1 - q$ is a constant < 1, then convergence is polynomial, but if the margin q from 1 is very small, inverse exponential in the size of the input I (as is generally the case, for example in Shapley's game), then convergence is slow.

For a monotone function F_I for which we want to compute the least fixed point, as in many of the examples above (stochastic games, branching processes, RMC etc.), we can start from $x_0 = 0$ (which is lower than the LFP) and apply repeatedly F_I; the process will converge to the desired LFP, but again convergence is generally slow. Note that for many of these problems, obtaining a weak ϵ-approximation for ϵ constant or even inverse polynomial, $|I|^{-c}$ is easy: for example, in a simple stochastic game or a branching process, the vector x is bounded from above by the all-1 vector and $F_I^k(0), k = 0, 1, 2, \ldots$ increases monotonically with k, so after at most n/ϵ iterations we will get a weak ϵ-approximate fixed point x. However, such a point x is of no use in estimating the actual values or probabilities that we want to compute. Approximating the value of a simple stochastic game even within additive error $1/2$ is an open problem.

For general Brouwer functions F we cannot simply apply iteratively F from some starting point x_0 and hope to converge to a fixed point. There is extensive algorithmic work on the approximate computation of Brouwer fixed points, starting with Scarf's fundamental algorithm [56], The standard proof of Brouwer's theorem involves a combinatorial lemma, Sperner's lemma, combined with a (generally nonconstructive) compactness argument. Scarf's algorithm solves constructively Sperner's problem, and computes a weak ϵ-approximate fixed point for the function. Briefly, it works as follows. Assume wlog that the domain is the unit simplex $\Delta_n = \{x \geq 0 | \sum_i x_i = 1\}$, and consider a simplicial subdivision of Δ_n into simplices of sufficiently small diameter δ, so that $|x - y| \leq \delta$ implies $|F(x) - F(y)| \leq \epsilon/n$. Label ("color") each vertex v of the subdivision by an index $i = 1, \ldots, n$ such that $v_i > F_i(v)$; if v is not a fixed point there is at least one such index, if v is a fixed point then label v with say $\arg_i \max(v_i)$. Note that the unit vectors e_i at the n corners of the simplex Δ_n are labelled i, and all vertices on the facet $x_j = 0$ are labelled with an index $\neq j$. Sperner's lemma implies then that the subdivision has at least one panchromatic simplex, i.e. a small simplex S whose vertices have distinct labels. From the definition of the labels and the choice of δ it follows that any point $x \in S$ satisfies $|F(x) - x| \leq \epsilon$. Scarf's algorithm starts with a suitable subdivision and a boundary simplex whose vertices have $n - 1$ distinct indices (all except one), and then keeps moving to an adjacent simplex through the face with the $n - 1$ indices; the process cannot repeat any simplex of the subdivision, so it will end up at a panchromatic simplex S. Note that S may not contain any actual fixed points, and in fact may be located far from all of them, but any point x of S is a weak ϵ-approximation. If we take finer and finer subdivisions letting the diameter δ go down to 0, then the resulting sequence of weakly approximate fixed points must contain (by compactness) a subsequence that converges to a point, which must be a fixed point; this latter part however is nonconstructive in general.

There are several other subsequent methods for computing (approximate) fixed points, e.g. Newton-based, and homotopy methods (some of these assume differentiability and use also the derivatives of the function). Scarf's algorithm, as well other general-purpose algorithms, treat the function F as a black box. Such black box algorithms must take exponential time in the worst case to compute a weak approximation [33]. Furthermore, for strong approximation no finite amount of time is enough in the black box model [60], and there are also noncomputability results for computing equilibria and fixed points for a model where the function is given via a Turing machine [40, 53]. However, the restriction to black box access is a severe one, and the results do not mean that any of the specific problems we want to solve (for example, Nash equilibria) is necessarily hard.

4. Rational equilibria, Piecewise Linear Functions and the Class PPAD

Consider a 2-player finite game, with the payoffs given explicitly in terms of the two payoff matrices A_1, A_2 of the two players (i.e., the game is presented in *normal form*). Computing a specific Nash equilibrium, such as one that maximizes the payoff to one of the players, or to all the players, is NP-hard [30]. However, the search problem that asks for *any* Nash equilibrium is a different, 'easier' problem, and is unlikely to be NP-hard.

The 2-player case of the Nash equilibrium problem can be viewed either as a continuous or as a discrete problem, like Linear Programming: We can consider LP either as having a continuous solution space, namely all the real-valued points in the feasible polyhedron, or as having a discrete solution space, namely the vertices of the polyhedron or the feasible bases. Similarly, for 2-player games which correspond to a Linear Complementarity problem. A mixed strategy profile is a Nash equilibrium iff every pure strategy of each player is either at 0 level (not in the support) or is a best response to the strategy of the other player. Assumming the game is nondegenerate (we can always ensure this by a small perturbation) the supports of the mixed strategies determine uniquely the equilibrium: we can set up and solve a linear system of equations which equate the payoffs of the pure strategies in the support of each player, and check that the solution satisfies the appropriate inequalities for the pure strategies that are not in the supports. One consequence of this is that if the payoffs are rational then there are rational equilibria, of polynomial bit complexity in the input size, and they can be computed exactly. A second consequence is that Nash's theorem in this case can be proved directly, without resorting to a fixed point theorem, and algorithmically, namely by the Lemke-Howson algorithm [42]. The algorithm has similar flavor to Scarf's algorithm for fixed points. Mixed profiles can be labelled ('colored') by the set of pure strategies that are not in the support or that are best responses to the other player's strategy. The equilibria are the mixed profiles that are panchromatic, i.e., labeled with all the pure strategies of both players. Briefly, the algorithm starts from an artificial point that has all the colors except one, and then follows a path through a sequence of LP-like pivots, until it arrives at a panchromatic point (profile), which must be an equilibrium; the algorithm cannot repeat any point, because at any point there are only two possible pivots, one forward and one backward, and there is a finite number of points (supports) so it terminates. It is known that the algorithm takes exponential time in the worst case [55].

Papadimitriou defined in [49] a complexity class, PPAD, that captures the basic principles of these path-following algorithms: There is a finite number of candidate solutions, and an underlying directed graph of moves between the solutions where each solution has at most one forward and one backward move, i.e., the graph consists of a set of directed paths, cycles and isolated nodes; a source of one path is an artificial starting solution, and every other endpoint (source or sink) of every path is an answer to the problem (eg., an equilibrium). Formally, a search problem Π is in PPAD if each instance I has a set $S(I)$ of solutions which are (strings) polynomially bounded in the input size $|I|$, and there are polynomial-time algorithms for the following tasks: (a) test whether a given string I is an instance of Π and if so compute a initial solution s_0 in $S(I)$, (b) given I, s, test whether $s \in S(I)$ and if so compute a successor $succ_I(s) \in S(I)$ and a predecessor $pred_I(s) \in S(I)$, such that $pred_I(s_0) = s_0$, $succ_I(s_0) \neq s_0$, and $pred_I(succ_I(s_0)) = s_0$. The *pred* and *succ* functions induce a directed graph $G = (S(I), E)$, where $E = \{(u, v) | u \neq v, succ_I(u) = v, pred_I(v) = u\}$, and the answer set to the instance I of the search problem, $Ans(I)$, is the set of nodes of G, other than s_0 that have indegree + outdegree = 1, i.e., are endpoints of the paths; note that

$Ans(I) \neq \emptyset$ because there must be at least one more endpoint besides s_0. As is customary, the class is closed under polynomial-time reduction, i.e., if a search problem Π' reduces to a problem Π that satisfies the above definition, then Π' is considered also to belong to PPAD. Papadimitriou defined two other variants of this class in [49], PPA in which the underlying graph is undirected, and PPADS in which the graph is directed and the answer set consists only of the sinks of the paths. However, PPAD is the more interesting and richer of these classes in terms of natural problems.

The class PPAD lies somewhere between P and TFNP: all search problems in PPAD are total, and furthermore, for a given instance I, we can guess a solution s and verify that it is an answer. Thus, as in the case of PLS, problems in PPAD cannot be NP-hard unless NP=coNP.

By virtue of the Lemke-Howson algorithm, the Nash equilibrium problem for 2-player (normal form) games is in PPAD. For 3 or more players we cannot say that the Nash problem is in PPAD; for one thing the equilibria are irrational. But the following approximate ϵ-Nash version is in PPAD [15]. An ϵ-Nash equilibrium of a game is a (mixed) strategy profile such that no player can improve its payoff by more than ϵ by switching unilaterally to another strategy. (Note, this is not the same as being ϵ-close to a Nash equilibrium.) The ϵ-Nash problem is: given a normal form game Γ (with rational payoffs) and a rational $\epsilon > 0$, compute an ϵ-Nash equilibrium of Γ. (Note that ϵ is given as usual in binary, so polynomial time means polynomial in $|\Gamma|$ and $log(1/\epsilon)$.) The complexity of the Nash problem was one of the main motivations for the original introduction of PPAD. A recent sequence of papers culminated in showing that the Nash equilibrium problem for 2-player games is PPAD-complete [15, 8], that is, if the problem can be solved in polynomial time, then so can all the problems in PPAD. Furthermore, even the ϵ-Nash equilibrium problem for ϵ inverse polynomial, i.e. even with ϵ given in unary, is also PPAD-complete for 2-player games [11]. For all constant ϵ, an ϵ-Nash equilibrium can be computed in quasipolynomial time [43].

Another basic PPAD-complete problem is (a formalization of) the Sperner problem. The 2D case concerns the unit simplex (triangle) Δ_3 and its simplicial subdivision (i.e., triangulation) with vertices $v = (i_1/n, i_2/n, i_3/n)$ with $i_1 + i_2 + i_3 = n$. The 2D Sperner problem is as follows. The input consists of a number n in binary and a Boolean circuit which takes as input three natural numbers i_1, i_2, i_3 with $i_1 + i_2 + i_3 = n$ and outputs a color $c \in \{1, 2, 3\}$, with the restriction that $i_c \neq 0$. The problem is to find a trichromatic triangle, i.e. three vertices (triples) with pairwise distances $1/n$ that have distinct colors. The problem is in PPAD by Scarf's algorithm. The 3D Sperner problem was shown PPAD-complete in [49], and the 2D case was shown complete in [9].

The original paper showed PPAD-completeness also for a discretized version of Brouwer and related theorems (Kakutani, Borsuk-Ulam) in the style of the Sperner problem: a Brouwer function in 3D is given in terms of a binary number n (the resolution of a regular grid subdivision of the unit 3-cube) and a Boolean circuit that takes as input three natural numbers i_1, i_2, i_3 between 0 and n and outputs the value of the function at the point $(i_1/n, i_2/n, i_3/n)$. The function is then linearly interpolated in the rest of the unit cube according to a standard simplicial subdivision with the grid points as vertices. The paper showed also completeness for a discretized version of the market equilibrium problem for an exchange economy.

In [12] Codenoti et al. show PPAD-completeness of the price equilibrium for a restricted case of Leontief exchange economies, i.e. economies in which each agent i wants commodities

in proportion to a specified (nonnegative) vector $(a_{i1}, \ldots, a_{ik})$; that is, the utility function of agent i is $u_i(x) = \min\{x_{ij}/a_{ij} | j = 1, \ldots, k; a_{ij} \neq 0\}$. In general, such economies may not have an equilibrium, and it is NP-hard to determine if there is one [12]. However, a restricted subclass of Leontief economies has equilibria and is equivalent to the Nash equilibrium problem for 2-player games. This restricted Leontief class is as follows: the agents are partitioned into two groups, every agent brings one distinct commodity to the market, and agents in the first group want commodities only of agents in the second group and vice-versa.

The class PPAD cannot capture of course general Brouwer functions since many of them have irrational fixed points as we saw in the last section. (We could discretize such a function, but then the resulting approximating function has new fixed points, which may have no relation and can be very far from the fixed points of the original function.) However there is a natural class of functions that are guaranteed to have rational fixed points, which are in PPAD and in a sense characterize the class [26]. Consider the search problem Π of computing a fixed point for a family of Brouwer functions $\mathcal{F} = \{F_I | I \text{ an instance of } \Pi\}$. We say that Π is a *polynomial piecewise linear* problem if the following hold: For each instance I, the domain is divided by hyperplanes into polyhedral cells, the function F_I is linear in each cell and is of course continuous over the whole domain. The coefficients of the function in each cell and of the dividing hyperplanes are rationals of size bounded by a polynomial in $|I|$. These are not given explicitly in the input, in fact there may be exponentially many dividing hyperplanes and cells. Rather, there is an oracle algorithm that runs in time polynomial in $|I|$ which generates a sequence of queries of the form $ax \leq b$? adaptively (i.e., the next query depends on I and the sequence of previous answers), and at the end either outputs 'No' (i.e., x is not in the domain) or identifies the cell of x and outputs the coefficients c, c' of the function $F_I(x) = cx + c'$. As shown in [26], all polynomial piecewise linear problems are in PPAD (they all have rational fixed points of polynomial size). Examples include the simple stochastic games, the discretized Brouwer functions obtained from linear interpolation on a grid, and the Nash equilibrium problem for 2-player games (Nash's function in nonlinear even for 2 players, but there is another piecewise linear function whose fixed points are also exactly the Nash equilibria).

The class PPAD captures also the approximation in the weak ('almost') sense for a broad class of Brouwer functions, and in some cases also the strong approximation ('near') problem [26]. Consider a family of functions $\mathcal{F} = \{F_I\}$. We say $\mathcal{F}$ is *polynomially computable* if for every instance I and rational vector x in the domain, the image $F_I(x)$ is rational and can be computed in time polynomial in the size of I and of x. $\mathcal{F}$ is called *polynomially continuous* if there is a polynomial $q(z_1, z_2)$ such that for all instances I and all rational $\epsilon > 0$, there is a rational $\delta > 0$ such that $size(\delta) \leq q(|I|, size(\epsilon))$ and such that for all $x, y \in D_I$, $|x - y| < \delta \;\Rightarrow\; |F_I(x) - F_I(y)| < \epsilon$. If $\mathcal{F}$ is polynomially computable and polynomially continuous, then the weak approximation problem (given instance I and rational $\epsilon > 0$, compute a weakly ϵ-approximate fixed point of F_I) is in PPAD by virtue of Scarf's algorithm. Furthermore, if the functions F_I happen to be also contracting with contraction rate $< 1 - 2^{-poly(|I|)}$, then strong approximation reduces to weak approximation, and the strong approximation problem (given I, ϵ, compute a point x that is within ϵ of some fixed point x^* of F_I) is also in PPAD; Shapley's problem is an example that satisfies this condition. Moreover, if in addition the functions F_I have rational fixed points of polynomial size, then strongly ϵ-approximate fixed points with small enough ϵ can be

rounded to get exact fixed points, and thus the exact problem is in PPAD; simple stochastic games, perturbed with a small discount [13], are such an example.

5. Irrational Equilibria, Nonlinear Functions, and the Class FIXP

Games with 3 or more players are quite different from 2-player games: Nash equilibria are generally irrational; knowing the support of an equilibrium does not help us much, and there may be many different such equilibria. There are many search problems as we saw in Section 3, and in particular many problems that can be cast in a fixed point framework, where the objects that we want to compute (the answers) are irrational. Of course we cannot compute them exactly in the usual Turing machine model of computation. One can consider the exact computation and the complexity of such search problems in a real model of computation [6]. In the usual (discrete) Turing model of computation and complexity, we have to state carefully and precisely what is the (finite) information about the solution that we want to compute, as the nature of the desired information can actually affect the complexity of the problem, i.e., some things may be easier to compute than others. That is, from a search problem Π with a continuous solution space, another search problem Π' is derived with a discrete space. Several types of information are potentially of interest, leading to different problems Π'.

Consider for example Shapley's stochastic game. Some relevant questions about the value of the game are the following: (i) *Decision problem*: Given game Γ and rational r, is the value of the game $\geq r$?, (ii) *Partial computation*: Given Γ, integer k, compute the k most significant bits of the value, (iii) *Approximation*: Given Γ, rational $\epsilon > 0$, compute an ϵ approximation to the value. Similar questions can be posed about the optimal strategies of the players. The value of a game is a problem with a unique answer; for multivalued search problems (e.g., optimal strategy, Nash equilibrium etc.) care must be taken in the statement of the discrete problems (e.g., the decision problem) so that it does not become harder than the search problem itself; in general, the requirement in the multivalued case is that the response returned for the discrete problem should be valid for some answer to the continuous search problem. As we said in the previous section, the approximation problem for the value of Shapley's game is in PPAD (and it is open whether it is P). The decision (and partial computation) problem however seems to be harder and it is not at all clear that it is even in NP; in fact showing that it is in NP would answer a well-known longstanding open problem. The same applies to many other problems. The best upper bound we know for the decision (and partial computation) problem for Shapley's games and for many of the other fixed point problems listed in Section 3 (eg., branching processes, RMCs etc) is PSPACE.

The *Square Root Sum* problem (`Sqrt-Sum` for short) is the following problem: given positive integers $d_1, \ldots, d_n$ and k, decide whether $\sum_{i=1}^{n} \sqrt{d_i} \leq k$. This problem arises often for example in geometric computations, where the square root sum represents the sum of Euclidean distances between given pairs of points with integer (or rational) coordinates; for example, determining whether the length of a specific spanning tree, or a TSP tour of given points on the plane is bounded by a given threshold k amounts to answering such a problem. This problem is solvable in PSPACE, but it has been a major open problem since the 1970's (see, e.g., [28, 62]) whether it is solvable even in NP (or better yet, in P). A related, and in a sense more powerful and fundamental, problem is the `PosSLP` *problem*: given a division-free straight-line program, or equivalently, an arithmetic circuit with operations $+, -, *$ and

inputs 0 and 1, and a designated output gate, determine whether the integer N that is the output of the circuit is positive. The importance of this problem was highlighted in [2], which showed that it is the key problem in understanding the computational power of the Blum-Shub-Smale model of real computation [6] using rational numbers as constants, in which all operations on rationals take unit time, no matter their size; importantly, integer division (the floor function) is not allowed (unit cost models with integer division or logical bit operations can solve in polynomial time all PSPACE problems, see e.g. [19] for an overview of machine models and references). This is a powerful model in which the `Sqrt-Sum` problem can be decided in polynomial time [62]). Allender et al. [2] showed that the set of discrete decision problems that can be solved in P-time in this model is equal to $P^{\texttt{PosSLP}}$, i.e. problems solvable in P using a subroutine for `PosSLP`. They showed also that `PosSLP` and `Sqrt-Sum` lie in the Counting Hierarchy (a hierarchy above PP).

The `Sqrt-Sum` problem can be reduced to the decision version of many problems: the Shapley problem [26], concurrent reachability games [25], branching processes, Recursive Markov chains [22], Nash equilibria for 3 or more players [26]. The `PosSLP` problem reduces also to several of these. Hence placing any of these problems in NP would imply the same for `Sqrt-Sum` and/or `PosSLP`. Furthermore, for several problems, the approximation of the desired objects is also at least as hard. In particular, approximating the termination probability of a Recursive Markov chain within any constant additive error < 1 is at least as hard as the `Sqrt-Sum` and the `PosSLP` problems [26].

A similar result holds for the approximation of Nash equilibria in games with 3 or more players. Suppose we want to estimate the probability with which a particular pure strategy, say strategy 1 of player 1, is played in a Nash equilibrium (any one); obviously, the value $1/2$ estimates it trivially with error $\leq 1/2$. Guaranteeing a constant error $< 1/2$ is at least as hard as the `Sqrt-Sum` and the `PosSLP` problems [26], i.e. it is hard to tell whether the strategy will be played with probability very close to 0 or 1.

The constructions illustrate also the difference between strong and weak approximate fixed points generally, and for specific problems in particular. Recall that for RMCs we can compute very easily a weak ϵ-approximate fixed point for any constant $\epsilon > 0$; however it is apparently much harder to obtain a strong approximation, i.e. approximate the actual probabilities within any nontrivial constant. In the RMC case the weak approximation is irrelevant. However, in the case of Nash equilibria, the weak approximation of Nash's function is also very natural and meaningful: it is essentially equivalent to the notion of ϵ-Nash equilibrium (there is a small polynomial change in ϵ in each direction). For every game Γ and $\epsilon > 0$, we can choose a δ of bit-size polynomial in the size of Γ and ϵ so that every strategy profile that is within distance δ of a Nash equilibrium is ϵ-Nash (i.e. all strongly approximate points are also weakly approximate with a 'small' change in ϵ). However, the converse is not true: For every n there is a 3-player game of size $O(n)$, with an ϵ-Nash equilibrium, x', where $\epsilon = 1/2^{2^{\Omega(n)}}$, such that x' has distance $1 - 2^{-poly}$ (i.e., almost 1) from every Nash equilibrium [26].

For 2-player games, as we said there is a direct, algorithmic proof of the existence of Nash equilibria (by the Lemke-Howson algorithm). But for 3 and more players, the only proofs known are through a fixpoint theorem (and there are several proofs known using different Brouwer functions or Kakutani's theorem). In [26] we defined a class of search problems, FIXP, that can be cast as fixed point problems of functions that use the usual algebraic operations and max, min, like Nash's function, and the other functions for the problems discussed in Section 3. Specifically, FIXP is the class of search problems Π, such

that there is a polynomial-time algorithm which, given an instance I, constructs an algebraic circuit (straight-line program) C_I over the basis $\{+, *, -, /, \max, \min\}$, with rational constants, that defines a continuous function F_I from a domain to itself (for simplicity, standardized to be the unit cube, other domains can be embedded into it), with the property that $Ans_\Pi(I)$ is the set of fixed points of F_I. The class is closed as usual under reductions. In the usual case of discrete search problems, a reduction from problem A to problem B consists of two polynomial-time computable functions, a function f that maps instances I of A to instances $f(I)$ of B, and a second function g that maps solutions y of the instance $f(I)$ of B to solutions x of the instance I of A. The difference here is that the solutions are real-valued, not discrete, so we have to specify what kind of functions g are allowed. It is sufficient to restrict the reverse function g to have a particularly simple form: a separable linear transformation with polynomial-time computable rational coefficients; that is, $x = g(y)$, where each $g_i(y)$ is of the form $a_i y_j + b_i$ for some j, where a_i, b_i are rationals computable from I in polynomial time. Examples of problems in FIXP include: Nash equilibrium for normal form games with any number of players, price equilibrium in exchange economies with excess demand functions given by algebraic formulas or circuits, the value (and optimal strategies) for Shapley's stochastic games, extinction probabilitites of branching processes, and probability of languages generated by stochastic context-free grammars.

FIXP is a class of search problems with continuous solution spaces, and corresponding to each such problem Π, there are the associated discrete problems: decision, approximation etc. All the accociated discrete problems can be expressed in the existential theory of the reals, and thus, using decision procedures for this theory [7, 52], it follows that they are all in PSPACE. As we mentioned, many of these problems are at least as hard as the `Sqrt-Sum` and the `PosSLP` problems, for which the current best upper bounds are barely below PSPACE. On the other hand, we do not know of any lower bounds, so in principle they could all be in P (though this is very doubtful). The Nash equilibrium problem for 3 players is complete for FIXP; it is complete in all senses, e.g., its approximation problem is as hard as the approximation of any other FIXP problem, the decision problem is at least as hard as the decision problem for any problem in FIXP, etc. [26]. The price equilibrium problem for algebraic excess demand functions is another complete problem.

A consequence of the completeness results is that the class FIXP stays the same under several variations. For example, using formulas instead of circuits in the representation of the functions does not affect the class (because Nash's function is given by a formula). Also, FIXP stays the same if we use circuits over $\{+, *, \max\}$ and rational constants (i.e., no division), because there is another function whose fixed points are also the Nash equilibria, and which can be implemented without division [26].

Of course FIXP contains PPAD, since it contains its complete problems, for example 2-player Nash. Actually, the piecewise linear fragment of FIXP corresponds exactly to PPAD. Let *Linear-FIXP* be the class of problems that can be expressed as (reduced to) exact fixed point problems for functions given by algebraic circuits using $\{+, -, \max, \min\}$ (equivalently, {+,max}) and multiplication with rational constants only; no division or multiplications of two gates/inputs is allowed. Then Linear-FIXP is equal to PPAD.

In several problems, we want a particular fixed point of a system $x = F(x)$, not just any one. In particular, in several of the probems discussed in Section 3 for example, the function F is a monotone operator and we want a Least Fixed Point. To place such a problem in FIXP, one has to restrict the domain in a suitable, but polynomial, way so that

only the desired fixed point is left in the domain. For some problems, we know how to do this (for example, extinction probabilities of branching processes), but for others (e.g. recursive Markov chains) it is not clear that this can be done in polynomial time. In any case, the paradigm of a LFP of a monotone operator is one that appears in many common settings, and which deserves its own separate treatment.

6. Conclusions

Many problems, from a broad, diverse range of areas, involve the computation of an equilibrium or fixed point of some kind. There is a long line of research (both mathematical and algorithmic) in each of these areas, but for many of these basic problems we still do not have polynomial time algorithms, nor do we have hard evidence of intractability (such as NP-hardness). We reviewed a number of such problems here, and we discussed three complexity classes, PLS, PPAD and FIXP, that capture essential aspects of several types of such problems. The classes PLS and PPAD lie somewhere between P and TFNP (total search problems in NP), and FIXP (more precisely, the associated discrete problems) lie between P and PSPACE. These, and the obvious containment PPAD $\subseteq$ FIXP, are the only relationships we currently know between these classes and the other standard complexity classes. It would be very interesting and important to improve on this state of knowledge. Furthermore, there are several important problems that are in these classes, but are not (known to be) complete, so it is possible that one can make progress on them, without resolving the relation of the classes themselves.

References

[1] H. Ackermann, H. Roglin, B. Vocking. On the impact of combinatorial structure on congestion games. *Proc. 47th IEEE FOCS*, 2006.

[2] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *Proc. 21st IEEE Comp. Compl. Conf.*, 2006.

[3] K.J. Arrow, G. Debreu. Existence of an equilibrium for a competitive economy. *Econometrica*, 22, pp. 265-290, 1954.

[4] R. M. Anderson. "Almost" implies "Near". *Trans. Am. Math. Soc.*, 296, pp. 229-237, 1986.

[5] R.J. Aumann, S. Hart (eds.). *Handbook of Game Theory*, vol. 3, North-Holland, 2002.

[6] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[7] J. Canny. Some algebraic and geometric computations in PSPACE. *Proc. ACM STOC*, pp. 460–467, 1988.

[8] X. Chen and X. Deng. Settling the complexity of two-player Nash equilibrium. *Proc. 47th IEEE FOCS*, pp. 261–272, 2006.

[9] X. Chen, X. Deng. On the complexity of 2d discrete fixed point problem. *Proc. ICALP*, pp. 489-599, 2006.

[10] X. Chen, X. Deng. On algorithms for discrete and approximate Brouwer fixed points. *Proc. ACM STOC*, pp. 323-330, 2005.

[11] X. Chen, X. Deng, and S. H. Teng. Computing Nash equilibria: approximation and smoothed complexity. *Proc. 47th IEEE FOCS*, pp. 603–612, 2006.

[12] B. Codenotti, A. Saberi, K. Varadarajan, Y. Ye. Leontieff economies encode nonzero sum two-player games. *Proc. SIAM SODA*, pp. 659-667, 2006.

[13] A. Condon. The complexity of stochastic games. *Inf. & Comp.*, 96(2):203–224, 1992.

[14] C. Daskalakis, A. Fabrikant, and C. Papadimitriou. The game world is flat: The complexity of Nash equilibria in succinct games. *Proc. ICALP*, 2006.

[15] C. Daskalakis, P. Goldberg, and C. Papadimitriou. The complexity of computing a Nash equilibrium. *Proc. ACM STOC*, pp. 71–78, 2006.
[16] L. de Alfaro, T. A. Henzinger, O. Kupferman. Concurrent reachability games. *Proc. IEEE FOCS*, pp. 564-575, 1998.
[17] G. Debreu. Excess demand functions. *J. Math. Econ.* 1, pp. 15-21, 1974.
[18] A. Ehrenfeucht, J. Mycielski. Positional strategies for mean payoff games. *Intl. J. Game Theory*, 8, pp. 109-113, 1979.
[19] P. van Emde Boas. Machine models and simulations. In *Handbook of Theoretical Computer Science*, vol. A, J. van Leeuwen ed., MIT Press, pp. 1–66, 1990.
[20] E. A. Emerson, C. Jutla. Tree automata, μ-calculus and determinacy. *Proc. IEEE FOCS*, pp. 368-377, 1991.
[21] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. *Proc. of 19th IEEE LICS'04*, 2004.
[22] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of non-linear equations. *Proc. STACS*, 2005. (Full expanded version available from `http://homepages.inf.ed.ac.uk/kousha`).
[23] K. Etessami and M. Yannakakis. Recursive Markov decision processes and recursive stochastic games. *Proc. 32nd ICALP*, 2005.
[24] K. Etessami and M. Yannakakis. Efficient qualitative analysis of classes of recursive Markov decision processes and simple stochastic games. *Proc. 23rd STACS*, Springer, 2006.
[25] K. Etessami and M. Yannakakis. Recursive concurrent stochastic games. *Proc. 33rd ICALP*, 2006.
[26] K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points. *Proc. IEEE FOCS*, 2007.
[27] A. Fabrikant, C.H. Papadimitriou, K. Talwar. The complexity of pure Nash equilibria. *Proc. ACM STOC*, pp. 604-612, 2004.
[28] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. *Proc. 8th ACM STOC*, pp. 10–22, 1976.
[29] J. Geanakoplos. Nash and Walras equilibrium via Brouwer. *Economic Theory*, 21:585–603, 2003.
[30] I. Gilboa and E. Zemel. Nash and correlated equilibria: some complexity considerations. *Games and Economic Behavior*, 1:80–93, 1989.
[31] P. Haccou, P. Jagers, V. A Vatutin. *Branching Processes: Variation, Growth, and Extinction of Populations*. Cambridge U. Press, 2005.
[32] T. E. Harris. *The Theory of Branching Processes*. Springer-Verlag, 1963.
[33] M. D. Hirsch, C. H. Papadimitriou, S. A. Vavasis. Exponential lower bounds for finding Brouwer fixed points. *J. Complexity*, 5, pp. 379-416, 1989.
[34] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci.* 79, pp. 2554-2558, 1982.
[35] D. S. Johnson. The NP-completeness column: Finding needles in haystacks. *ACM Trans. Algorithms* 3, 2007.
[36] D. S. Johnson, C. H. Papadimitriou, M. Yannakakis. How easy is local search? *J. Comp. Sys. Sci.*, 37, pp. 79-100, 1988.
[37] B. Juba. On the hardness of simple stochastic games. Master's thesis, CMU, 2005.
[38] M. Jurdzinski. Deciding the winner in parity games is in UP$\cap$coUP. *Inf. Proc. Let.* 68, pp. 119-124, 1998.
[39] M. Kearns. Graphical games. In [48], pp. 159–180, 2007.
[40] K.-I. Ko. Computational complexity of fixpoints and intersection points. *J. Complexity*, 11, pp. 265-292, 1995.
[41] A. N. Kolmogorov and B. A. Sevastyanov. The calculation of final probabilities for branching random processes. *Doklady*, 56:783–786, 1947. (Russian).
[42] C. Lemke, J. Howson. Equilibrium points of bimatrix games. *J. SIAM*, pp. 413-423, 1964.
[43] R.J. Lipton, E. Markakis, A. Mehta. Playing large games using simple strategies. *Proc. ACM Conf. Elec. Comm.*, 36-41, 2003.
[44] R.J. Lipton, E. Markakis. Nash equilibria via polynomial equations. *Proc. LATIN*, 2004.
[45] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[46] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54:289–295, 1951.
[47] A. Neyman and S. Sorin, eds. *Stochastic Games and Applications*. Kluwer, 2003.
[48] N. Nisan, T. Roughgarden, E. Tardos, V. Vazirani. *Algorithmic Game Theory*. Cambridge Univ. Press, 2007.
[49] C. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.
[50] C. Papadimitriou. The complexity of finding Nash equilibria. In [48], pp. 29-52, 2007.
[51] A. Puri. Theory of hybrid systems and discrete event systems. PhD Thesis, UC Berkeley, 1995.
[52] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, parts I-III. *J. Symb. Comp.*, 13(3):255–352, 1992.
[53] M. Richter, K.-C. Wong. Non-computability of competitive equilibrium. *Economic Theory*, 14, pp. 1-27, 1999.
[54] R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *Intl. J. Game Theory* 2, pp. 65-67, 1973.
[55] R. Savani, B. von Stengel. Hard to solve bimatrix games. *Econometrica* 74, pp. 397-429, 2006.
[56] H. Scarf. The approximation of fixed points of a continuous mapping. *SIAM J. Appl. Math.*, 15:1328–1343, 1967.
[57] H. Scarf. *The Computation of Economic Equilibria*. Yale University Press, 1973.
[58] A. Schaffer, M. Yannakakis. Simple Local Search Problems that are Hard to Solve. *SIAM J. Comp.*, 20, pp. 56-87, 1991.
[59] L.S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci.*, 39:1095–1100, 1953.
[60] K. Sikorski. *Optimal solution of nonlinear equations*, Oxford Univ. Press, 2001.
[61] B. von Stengel. Computing equilibria for two-person games. In [5], pp. 1723-1759, 2002.
[62] P. Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *J. of Complexity*, pp. 393–397, 1992.
[63] H. Uzawa. Walras' existence theorem and Brouwer's fixpoint theorem. *Econ. Stud. Quart.*, 13, pp. 59-62, 1962.
[64] B. Vöcking. Congestion Games: Optimization in Competition. *Proc. 2nd ACiD*, pp. 9-20, 2006.
[65] M. Yannakakis. The analysis of local search problems and their heuristics. *Proc. STACS*, pp. 298-311, 1990.
[66] M. Yannakakis. Computational complexity of local search. In *Local Search in Combinatorial Optimization*, E.H.L. Aarts, J.K. Lenstra eds., John Wiley, 1997.
[67] U. Zwick, M. S. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158, pp. 343-359, 1996.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 39-48
www.stacs-conf.org

PUSHDOWN COMPRESSION

P. ALBERT [1], E. MAYORDOMO [1], P. MOSER [2], AND S. PERIFEL [3]

[1] Dept. de Informática e Ingeniería de Sistemas , Universidad de Zaragoza. Edificio Ada Byron, María de Luna 1 - E-50018 Zaragoza (Spain)
E-mail address: {mpalbert,elvira}@unizar.es

[2] Dept. of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland
E-mail address: pmoser@cs.nuim.ie

[3] LIP, École Normale Supérieure de Lyon. UMR 5668 ENS Lyon, CNRS, UCBL, INRIA
E-mail address: asylvain.perifel@ens-lyon.fr

Abstract. The pressing need for efficient compression schemes for XML documents has recently been focused on stack computation [6, 9], and in particular calls for a formulation of information-lossless stack or pushdown compressors that allows a formal analysis of their performance and a more ambitious use of the stack in XML compression, where so far it is mainly connected to parsing mechanisms. In this paper we introduce the model of pushdown compressor, based on pushdown transducers that compute a single injective function while keeping the widest generality regarding stack computation.

The celebrated Lempel-Ziv algorithm LZ78 [10] was introduced as a general purpose compression algorithm that outperforms finite-state compressors on all sequences. We compare the performance of the Lempel-Ziv algorithm with that of the pushdown compressors, or compression algorithms that can be implemented with a pushdown transducer. This comparison is made without any a priori assumption on the data's source and considering the asymptotic compression ratio for infinite sequences. We prove that Lempel-Ziv is incomparable with pushdown compressors.

1. Introduction

The celebrated result of Lempel and Ziv [10] that their algorithm is asymptotically better than any finite-state compressor is one of the major theoretical justifications of this widely used algorithm. However, until recently the natural extension of finite-state to pushdown compressors has received much less attention, a situation that has changed due to new specialized compressors.

Key words and phrases: Finite-state compression, Lempel-Ziv algorithm, pumping-lemma, pushdown compression, XML document.

Research supported in part by Spanish Government MEC Project TIN 2005-08832-C03-02 and by Aragón Government Dept. Ciencia, Tecnología y Universidad, subvención destinada a la formación de personal investigador-B068/2006.

In particular, XML is rapidly becoming a standard for the creation and parsing of documents, however, a significant disadvantage is document size, even more since present day XML databases are massive. Since 1999 the design of new compression schemes for XML is an active area where the use of syntax directed compression is specially adequate, that is, compression centered on the grammar-based generation of XML-texts and performed with stack memory [6, 9].

On the other hand the work done on stack transducers has been basic and very connected to parsing mechanisms. Transducers were initially considered by Ginsburg and Rose in [4] for language generation, further corrected in [5], and summarized in [1]. For these models the role of nondeterminism is specially useful in the concept of λ-rule, that is a transition in which a symbol is popped from the stack without reading any input symbol.

In this paper we introduce the concept of pushdown compressor as the most general stack transducer that is compatible with information-lossless compression. We allow the full power of λ-rules while having a deterministic (unambiguous) model. The existence of endmarkers is discussed, since it allows the compressor to move away from mere prefix extension by exploiting λ-rules.

The widely-used Lempel-Ziv algorithm LZ78 [10] was introduced as a general purpose compression algorithm that outperforms finite-sate compressors on all sequences when considering the asymptotic compression ratio. This means that for infinite sequences, the algorithm attains the (a posteriori) finite state or block entropy. If we consider an ergodic source, the Lempel-Ziv compression coincides exactly with the entropy of the source with high probability on finite inputs. This second result is useful when the data source is known, whereas it is not informative for arbitrary inputs. We don't know the performance of Lempel-Ziv on individual long or infinite inputs (notice that an infinite sequence is Lempel-Ziv incompressible with probability one). For the comparison of compression algorithms on general sequences, either an experimental or a formal approach is needed, such as that used in [8]. In this paper we follow [8] using a worse case approach, that is, we consider asymptotic performance on every infinite sequence.

We compare the performance of the Lempel-Ziv algorithm with that of the pushdown-compressors, or compression algorithms that can be implemented with a pushdown transducer. This comparison is made without any a priori assumption on the data's source and considering the asymptotic compression ratio for infinite sequences.

We prove that Lempel-Ziv compresses optimally a sequence that no pushdown transducer compresses at all, that is, the Lempel-Ziv and pushdown compression ratios of this sequence are 0 and 1, respectively. For this result, we develop a powerful nontrivial pumping-lemma, that has independent interest since it deals with families of pushdown transducers, while known pumping-lemmas are restricted to recognizing devices [1].

In fact, Lempel-Ziv and pushdown compressing algorithms are incomparable, since we construct a sequence that is very close to being Lempel-Ziv incompressible while the pushdown compression ratio is at most one half. While Lempel-Ziv is universal for finite-state compressors, our theorem implies a strong non-universality result for Lempel-Ziv and pushdown compressors.

The paper is organized as follows. Section 2 contains some preliminaries. In section 3, we present our model of pushdown compressor with its basic properties and notation. In section 4 we show that there is a sequence on which Lempel-Ziv outperforms pushdown compressors and in section 5 we show that Lempel-Ziv and pushdown compression are incomparable. We

finish with a brief discussion of connections and consequences of these results for dimension and prediction algorithms.

2. Preliminaries

We write $\mathbb{Z}$ for the set of all integers, $\mathbb{N}$ for the set of all nonnegative integers and $\mathbb{Z}^+$ for the set of all positive integers. Let Σ be a finite alphabet, with $|\Sigma| \geq 2$. Σ^* denotes the set of finite strings, and Σ^∞ the set of infinite sequences. We write $|w|$ for the length of a string w in Σ^*. The empty string is denoted by λ. For $S \in \Sigma^\infty$ and $i, j \in \mathbb{N}$, we write $S[i..j]$ for the string consisting of the i^{th} through j^{th} bits of S, with the convention that $S[i..j] = \lambda$ if $i > j$, and $S[0]$ is the leftmost bit of S. We write $S[i]$ for $S[i..i]$ (the i^{th} bit of S). For $w \in \Sigma^*$ and $S \in \Sigma^\infty$, we write $w \sqsubseteq S$ if w is a prefix of S, i.e., if $w = S[0..|w| - 1]$. Unless otherwise specified, logarithms are taken in base $|\Sigma|$. For a string x, x^{-1} denotes x written in reverse order. We use $f(x) = \perp$ to denote that function f is undefined on x.

Let us give a brief description of the Lempel-Ziv (LZ) algorithm [10]. Given an input $x \in \Sigma^*$, LZ parses x in different phrases x_i, i.e., $x = x_1 x_2 \dots x_n$ ($x_i \in \Sigma^*$) such that every prefix $y \sqsubset x_i$, appears before x_i in the parsing (i.e. there exists $j < i$ s.t. $x_j = y$). Therefore for every i, $x_i = x_{l(i)} b_i$ for $l(i) < i$ and $b_i \in \Sigma$. We sometimes denote the number of phrases in the parsing of x as $P(x)$. After step i of the algorithm, the i first phrases $x_1, \dots, x_i$ have been parsed and stored in what we will call the *dictionary*. Thus, each step adds one word to the dictionary.

LZ encodes x_i by a prefix free encoding of $x_{l(i)}$ and the symbol b_i, that is, if $x = x_1 x_2 \dots x_n$ as before, the output of LZ on input x is

$$LZ(x) = c_{l(1)} b_1 c_{l(2)} b_2 \dots c_{l(n)} b_n$$

where c_i is a prefix-free coding of i (and $x_0 = \lambda$).

LZ is usually restricted to the binary alphabet, but the description above is valid for any Σ.

For a sequence $S \in \Sigma^\infty$, the LZ infinitely often compression ratio is given by

$$\rho_{LZ}(S) = \liminf_{n \to \infty} \frac{|LZ(S[0 \dots n-1])|}{n \log_2(|\Sigma|)}.$$

$\rho_{LZ}(S)$ corresponds to the best-case performance of Lempel-Ziv on finite prefixes of sequence S.

We also consider the almost everywhere compression ratio

$$R_{LZ}(S) = \limsup_{n \to \infty} \frac{|LZ(S[0 \dots n-1])|}{n \log_2(|\Sigma|)}.$$

$R_{LZ}(S)$ corresponds to the worst-case performance of Lempel-Ziv on finite prefixes of sequence S.

3. Pushdown compression

Definition 3.1. A *pushdown compressor* (PDC) is a 7-tuple

$$C = (Q, \Sigma, \Gamma, \delta, \nu, q_0, z_0)$$

where

- Σ is the finite input alphabet

- Q is a finite set of states
- Γ is the finite stack alphabet
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$ is the transition function
- $\nu : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to \Sigma^*$ is the output function
- $q_0 \in Q$ is the initial state
- $z_0 \in \Gamma$ is the start stack symbol

We use δ_Q and δ_{Γ^*} for the projections of function δ. Note that the transition function δ accepts λ as an input character in addition to elements of Σ, which means that C has the option of not reading an input character while altering the stack. In this case $\delta(q, \lambda, a) = (q', \lambda)$, that is, we pop the top symbol of the stack. To enforce determinism, we require that at least one of the following hold for all $q \in Q$ and $a \in \Gamma$:

- $\delta(q, \lambda, a) = \perp$
- $\delta(q, b, a) = \perp$ for all $b \in \Sigma$

We restrict δ so that z_0 cannot be removed from the stack bottom, that is, for every $q \in Q$, $b \in \Sigma \cup \{\lambda\}$, either $\delta(q, b, z_0) = \perp$, or $\delta(q, b, z_0) = (q', vz_0)$, where $q' \in Q$ and $v \in \Gamma^*$.

There are several natural variants for the model of pushdown transducer [1], both allowing different degrees of nondeterminism and computing partial (multi)functions by requiring final state or empty stack termination conditions. Our purpose is to compute a total and well-defined (single valued) function in order to consider general-purpose, information-lossless compressors.

Notice that we have not required here or in what follows that the computation should be invertible by another pushdown transducer, which is a natural requirement for practical compression schemes. Nevertheless the unambiguity condition of a single computation per input gives as a natural upper bound on invertibility.

We first consider the transition function δ as having inputs in $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma^+$, meaning that only the top symbol of the stack is relevant. Then we use the extended transition function $\delta^* : Q \times \Sigma^* \times \Gamma^+ \to Q \times \Gamma^*$, defined recursively as follows. For $q \in Q$, $v \in \Gamma^+$, $w \in \Sigma^*$, and $b \in \Sigma$

$$\delta^*(q, \lambda, v) = \begin{cases} \delta^*(\delta_Q(q, \lambda, v), \lambda, \delta_{\Gamma^*}(q, \lambda, v)), & \text{if } \delta(q, \lambda, v) \neq \perp; \\ (q, v), & \text{otherwise.} \end{cases}$$

$$\delta^*(q, wb, v) = \begin{cases} \delta^*(\delta_Q(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v)), \lambda, \delta_{\Gamma^*}(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v))), \\ \qquad \text{if } \delta^*(q, w, v) \neq \perp \text{ and } \delta(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v)) \neq \perp; \\ \perp, \quad \text{otherwise.} \end{cases}$$

That is, λ-rules are inside the definition of δ^*. We abbreviate δ^* to δ, and $\delta(q_0, w, z_0)$ to $\delta(w)$. We define the *output* from state q on input $w \in \Sigma^*$ with $z \in \Gamma^*$ on the top of the stack by the recursion $\nu(q, \lambda, z) = \lambda$,

$$\nu(q, wb, z) = \nu(q, w, z)\nu(\delta_Q(q, w, z), b, \delta_{\Gamma^*}(q, w, z)).$$

The *output* of the compressor C on input $w \in \Sigma^*$ is the string $C(w) = \nu(q_0, w, z_0)$.

The input of an information-lossless compressor can be reconstructed from the output and the final state reached on that input.

Definition 3.2. A PDC $C = (Q, \Sigma, \Gamma, \delta, \nu, q_0, z_0)$ is *information-lossless* (IL) if the function

$$\Sigma^* \to \Sigma^* \times Q$$
$$w \to (C(w), \delta_Q(w))$$

is one-to-one. An *information-lossless pushdown compressor* ($ILPDC$) is a PDC that is IL.

Intuitively, a PDC *compresses* a string w if $|C(w)|$ is significantly less than $|w|$. Of course, if C is IL, then not all strings can be compressed. Our interest here is in the degree (if any) to which the prefixes of a given sequence $S \in \Sigma^\infty$ can be compressed by an ILPDC.

Definition 3.3. If C is a PDC and $S \in \Sigma^\infty$, then the *compression ratio* of C on S is

$$\rho_C(S) = \liminf_{n\to\infty} \frac{|C(S[0..n-1])|}{n \log_2(|\Sigma|)}$$

Definition 3.4. The *pushdown compression ratio* of a sequence $S \in \Sigma^\infty$ is

$$\rho_{PD}(S) = \inf\{\rho_C(S) \mid \text{ C is an ILPDC}\}$$

$\rho_{PD}(S)$ corresponds to the best-case performance of PD-compressors on S.

We can consider dual concepts R_C and R_{PD} by replacing lim inf with lim sup in the previous definition. $R_{PD}(S)$ corresponds to the worst-case performance of PD-compressors on S.

3.1. Endmarkers and pushdown compression

Two possibilities occur when dealing with transducers on finite words: should the end of the input be marked with a particular symbol # or not? As we will see, this is a rather subtle question. First remark that both approaches are natural: on the one hand, usual finite state or pushdown *acceptors* are one-way and do not know (and do not need to know) when they reach the end of the word; on the other hand, two-way finite state acceptors need to know it and everyday compression algorithms usually know (or at least are able to know) where the end of the input file takes place. For a word w, we will denote by $C(w)$ the output of a transducer C without endmarker, and $C(w\#)$ the output with an endmarker.

Unlike acceptors, transducers can take advantage of an endmarker: they can indeed output more symbols when they reach the end of the input word if it is marked with a particular symbol. This is therefore a more general model of transducers which, in particular, does not have the strong restriction of prefix extension: if there is no endmarker and C is a transducer, then for all words w_1, w_2, $w_1 \sqsubseteq w_2 \Rightarrow C(w_1) \sqsubseteq C(w_2)$. Let us see how this restriction limits the compression ratio.

Lemma 3.5. *Let C be an IL pushdown compressor with k states and working with no endmarker. Then on every word w of size $|w| \geq k$, the compression ratio of C is*

$$\frac{|C(w)|}{|w|} \geq \frac{1}{2k}.$$

Proof. Due to the injectivity condition, we can show that C has to output at least one symbol every k input symbols. Suppose on the contrary that there are words t, u, with $|u| = k$, such that C does not output any symbol when reading u on input $w = tu$. Then all the $k+1$ words t and $tu[0..i]$ for $0 \leq i \leq k-1$ have the same output by C, and furthermore two of them have the same final state because there are only k states. This contradicts injectivity. Thus C must output at least one symbol every k symbols, which proves the lemma. □

This limitation does not occur with endmarkers, as the following lemma shows.

Lemma 3.6. *For every k, there exists an IL pushdown compressor C with k states, working with endmarkers, such that the compression ratio of C on 0^n tends to $1/k^2$ when n tends to infinity, that is,*

$$\lim_{n\to\infty} \frac{|C(0^n)|}{n} = \frac{1}{k^2}.$$

Proof. (Sketch) On input 0^n, our compressor outputs (roughly) $0^{n/k^2}$ as follows: by selecting one symbol out of each k of the input word (counting modulo k thanks to k states), it pushes $0^{n/k}$ on the stack. Then at the end of the word, it pops the stack and outputs one symbol every k pop. Thus the output is $0^{n/k^2}$ (in fact, the remainder of n modulo k^2 also has to be taken into account).

To ensure injectivity, if the input word w is not of the form 0^n (that is, if it contains a 1), then C outputs $1w$. □

It is worth noticing that it is the injectivity condition that makes this computation impossible without endmarkers, because one cannot decide *a priori* whether the input word contains a 1. Thus pushdown compressors with endmarkers do not have the limitation of Lemma 3.5. Still, as Corollary 4.5 will show, pushdown compressors with endmarkers are not universal for finite state compressors, in the sense that a single pushdown compressor cannot be as good as any finite state compressor.

It is open whether pushdown compressors with endmarkers are strictly better than without, in the sense of the following question.

Open question. Do there exist an infinite sequence S, a constant $0 < \alpha \leq 1$ and an IL pushdown compressor C working with endmarkers, such that $\rho_C(S) < \alpha$, but $\rho_{C'}(S) \geq \alpha$, for every C' IL pushdown compressor working without endmarkers?

In the rest of the paper we consider both variants of compression: with and without endmarkers. We use the weakest variant for positive results and the strongest for negative ones, therefore showing stronger separations.

4. Lempel-Ziv outperforms Pushdown transducers

In this section we show the existence of an infinite sequence $S \in \{0,1\}^\infty$ compressible by Lempel-Ziv but not by pushdown compressors. More precisely, we show in Theorem 4.8 the following result: the almost everywhere Lempel-Ziv compression ratio on S is 0 but the infinitely often IL pushdown compression ratio is 1. Another (weaker) version will be stated in Theorem 4.9 for pushdown compressors with endmarkers.

The rough idea is that Lempel-Ziv compresses repetitions very well, whereas, if the repeated word is well chosen, pushdown compressors perform very poorly. We first show the claim on Lempel-Ziv and then prove a pumping-lemma for pushdown transducers in order to deal with the case of pushdown compressors.

4.1. Lempel-Ziv on periodic inputs

The sequence we will build consists of regions where the same pattern is repeated several times. This ensures that Lempel-Ziv algorithm compresses the sequence, as shown by the following lemmas.

We begin with finite words: Lempel-Ziv compresses well words of the form tu^n. The idea is that the dictionary remains small during the execution of the algorithm because there are few different subwords of same length in tu^n due to the period of size $|u|$. The statement is slightly more elaborated because we want to use it in the proof of Theorem 4.2 where we will need to consider the execution of Lempel-Ziv on a possibly nonempty dictionary.

Lemma 4.1. *Let $n \in \mathbb{N}$ and let $t, u, \in \Sigma^*$, where $u \neq \lambda$. Define $l = 1 + |t| + |u|$ and $w_n = tu^n$. Consider the execution of Lempel-Ziv on w_n starting from a dictionary containing $d \geq 0$ phrases. Then we have that*

$$\frac{|LZ(w_n)|}{|w_n|} \leq \frac{\sqrt{2l|w_n|}\log(d + \sqrt{2l|w_n|})}{|w_n|}.$$

This leads us to the following lemma on a particular infinite sequence.

Theorem 4.2 (LZ compressibility of repetitive sequences)**.** *Let $(t_i)_{i\geq 1}$ and $(u_i)_{i\geq 1}$ be sequences of words, where $u_i \neq \lambda, \forall i \geq 1$. Let $(n_i)_{i\geq 1}$ be a sequence of integers. Let S be the sequence defined by*

$$S = t_1 u_1^{n_1} t_2 u_2^{n_2} t_3 u_3^{n_3} \dots$$

If the sequence $(n_i)_{i\geq 1}$ grows sufficiently fast, then

$$R_{LZ}(S) = 0.$$

4.2. Pumping-lemma for injective pushdown transducers

This section is devoted to the statement and proof of a pumping-lemma for pushdown transducers. In the usual setting of recognition of formal languages by pushdown automata, the pumping-lemma comes from the equivalence between context-free grammars and pushdown automata, see for instance [11]. However, the proof is much less straightforward without grammars, as is our case since we deal with transducers and not acceptors. Moreover, there are three further difficulties: first, we have to consider what happens at the end of the word, after the endmarker (where the transducer can still output symbols when emptying the stack); second, we need a lowerbound on the size of the pumping part, that is, we need to pump on a sufficiently large part of the word; third, we need the lemma for an arbitrary finite family of automata, and not only one automaton. All this makes the statement and the proof much more involved than in the usual language-recognition framework. The proof consists in finding two similar configurations of the transducer so that we can repeat the input word read between them. The size of the input word has therefore to be large enough but note that in the following statement, this restriction is replaced by the possibility of pumping on an empty word u (as soon as $\alpha|w|^\beta < 1$ since we take the integer part).

Lemma 4.3 (Pumping-lemma)**.** *Let $\mathcal{F}$ be a finite family of ILPDC. There exist two constants $\alpha, \beta > 0$ such that $\forall w$, there exist $t, u, v \in \Sigma^*$ such that $w = tuv$ satisfying:*

- *$|u| \geq \lfloor \alpha |w|^\beta \rfloor$;*
- *$\forall C \in \mathcal{F}$, there exist two words x, y such that $C(tu^n) = xy^n$, $\forall n \in \mathbb{N}$.*

Taking into account endmarkers, we obtain the following corollary:

Corollary 4.4 (Pumping-lemma with endmarkers)**.** *Let $\mathcal{F}$ be a finite family of ILPDC. There exist two constants $\alpha, \beta > 0$ such that every word w can be cut in three pieces $w = tuv$ satisfying:*

(1) *$|u| \geq \lfloor \alpha |w|^\beta \rfloor$;*
(2) *there is an integer $c \geq 0$ such that for all $C \in \mathcal{F}$, there exist five words x, x', y, y', z such that for all $n \geq c$, $C(tu^n v\#) = xy^n z y'^{n-c} x'$.*

Let us state an immediate corollary concerning universality: pushdown compressors, even with endmarkers, cannot be universal for finite state compressors, in the sense that the compression ratio of a particular pushdown compressor cannot be always better than the compression ratio of every finite state compressor.

Corollary 4.5. *Let C be an IL pushdown compressor (with endmarkers). Then $\rho_C(0^\infty) > 0$. In particular, no pushdown compressor is universal for finite state compressors.*

Proof. By Corollary 4.4, there exist two integers k, k', ($k' \geq 1$), a constant $c \geq 0$ and five words x, x', y, y', z such that for all $n \geq c$, $C(0^k 0^{k'n} \#) = xy^n z y'^{n-c} x'$. By injectivity of C, y and y' cannot be both empty. Hence the size of the compression of $0^k 0^{k'n}$ is linear in n. This proves the first assertion.

Since for every $\epsilon > 0$ there exists an IL finite state compressor C' such that $R_{C'}(0^\infty) < \epsilon$, the pushdown compressor C cannot be universal for finite state compressors. □

4.3. A pushdown incompressible sequence

We now show that some sequences with repetitions cannot be compressed by pushdown compressors. We start by analyzing the performance of PDC on the factors of a Kolmogorov-random word (that is, a word w that contains at least $|w|$ bits of information in the (plain) Kolmogorov complexity sense, i.e. $K(w) \geq |w|$; or, to put it another way, a word that cannot be compressed by Turing machines). This result is valid even with endmarkers.

Lemma 4.6. *For every $\mathcal{F}$ finite family of $ILPDC$ with k states and for every constant $\epsilon > 0$, there exists $M_{\mathcal{F},\epsilon} \in \mathbb{N}$ such that, for any Kolmogorov random word $w = tu$, if $|u| \geq M_{\mathcal{F},\epsilon} \log |w|$ then the compression ratio for $C \in \mathcal{F}$ of u on input w is*

$$\frac{|C(tu)| - |C(t)|}{|u|} \geq 1 - \epsilon.$$

We can now build an infinite sequence of the form required in Theorem 4.2 that cannot be compressed by bounded pushdown automata. The idea of the proof is as follows: by Corollary 4.4, in any word w we can repeat a big part u of w while ensuring that the behaviour of the transducer on every copy of u is the same. If u is not compressible, the output will be of size almost $|u|$, therefore with a large number of repetitions the compression ratio is almost 1.

Theorem 4.7 (A pushdown incompressible repetitive sequence)**.** *Let Σ be a finite alphabet. There exist sequences of words $(t_k)_{k\geq 1}$ and $(u_k)_{k\geq 1}$, where $u_k \neq \lambda, \forall k \geq 1$, such that for every sequence of integers $(n_k)_{k\geq 1}$ growing sufficiently fast, the infinite string S defined by*

$$S = t_1 u_1^{n_1} t_2 u_2^{n_2} t_3 u_3^{n_3} \dots$$

verifies that

$$\rho_C(S) = 1,$$

$\forall C \in ILPDC$ (without endmarkers).

Combining it with Theorem 4.2 we obtain the main result of this section, there are sequences that Lempel-Ziv compresses optimally on almost every prefix, whereas no pushdown compresses them at all, even on infinitely many prefixes (Theorem 4.8) or using endmarkers (Theorem 4.9).

Theorem 4.8. *There exists a sequence S such that*

$$R_{LZ}(S) = 0$$

and

$$\rho_C(S) = 1$$

for any $C \in ILPDC$ (without endmarkers).

The situation with endmarkers is slightly more complicated, but using Corollary 4.4 (the pumping lemma with endmarkers) and a similar construction as Theorem 4.7 we obtain the following result. Note that we now use the limsup of the compression ratio for ILPDC with endmarkers.

Theorem 4.9. *There exists a sequence S such that*

$$R_{LZ}(S) = 0$$

and

$$R_C(S) = 1$$

for any $C \in ILPDC$ (using endmarkers).

5. Lempel-Ziv is not universal for Pushdown compressors

It is well known that LZ [10] yields a lower bound on the finite-state compression of a sequence [10], ie, LZ is universal for finite-state compressors.

The following result shows that this is not true for pushdown compression, in a strong sense: we construct a sequence S that is infinitely often incompressible by LZ, but that has almost everywhere pushdown compression ratio less than $\frac{1}{2}$.

Theorem 5.1. *For every $m \in \mathbb{N}$, there is a sequence $S \in \{0,1\}^\infty$ such that*

$$\rho_{LZ}(S) > 1 - \frac{1}{m}$$

and

$$R_{PD}(S) \leq \frac{1}{2}.$$

6. Conclusion

The equivalence of compression ratio, effective dimension, and log-loss unpredictability has been explored in different settings [2, 7, 13]. It is known that for the cases of finite-state, polynomial-space, recursive, and constructive resource-bounds, natural definitions of compression and dimension coincide, both in the case of infinitely often compression, related to effective versions of Hausdorff dimension, and that of almost everywhere compression, matched with packing dimension. The general matter of transformation of compressors in predictors and vice versa is widely studied [14].

In this paper we have done a complete comparison of pushdown compression and LZ-compression. It is straightforward to construct a prediction algorithm based on Lempel-Ziv compressor that uses similar computing resources, and it is clear that finite-state compression is always at least pushdown compression. This leaves us with the natural open question

of whether each pushdown compressor can be transformed into a pushdown prediction algorithm, for which the log-loss unpredictability coincides with the compression ratio of the initial compressor, that is, whether the natural concept of pushdown dimension defined in [3] coincides with pushdown compressibility. A positive answer would get pushdown computation closer to finite-state devices, and a negative one would make it closer to polynomial-time algorithms, for which the answer is likely to be negative [12].

Acknowledgement

The authors thank Victor Poupet for his help on the proof of Lemma 4.3.

References

[1] J. Autebert, J. Berstel, and L. Boasson. Context-free languages and pushdown automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, volume 1, Word, Language, Grammar*, pages 111–174. Springer-Verlag, 1997.

[2] J. J. Dai, J. I. Lathrop, J. H. Lutz, and E. Mayordomo. Finite-state dimension. *Theoretical Computer Science*, 310:1–33, 2004.

[3] D. Doty and J. Nichols. Pushdown dimension. *Theoretical Computer Science*, 381(1-3):105–123, 2007.

[4] S. Ginsburg and G. F. Rose. Preservation of languages by transducers. *Information and Control*, 9(2):153–176, 1966.

[5] S. Ginsburg and G. F. Rose. A note on preservation of languages by transducers. *Information and Control*, 12(5/6):549–552, 1968.

[6] S. Hariharan and P. Shankar. Evaluating the role of context in syntax directed compression of xml documents. In *Proceedings of the 2006 IEEE Data Compression Conference (DCC 2006)*, page 453, 2006.

[7] J. M. Hitchcock. *Effective Fractal Dimension: Foundations and Applications*. PhD thesis, Iowa State University, 2003.

[8] J. I. Lathrop and M. J. Strauss. A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data. In B. Carpentieri, editor, *Compression and Complexity of Sequences '97*, pages 123–135. IEEE Computer Society Press, 1998.

[9] C. League and K. Eng. Type-based compression of xml data. In *Proceedings of the 2007 IEEE Data Compression Conference (DCC 2007)*, pages 272–282, 2007.

[10] A. Lempel and J. Ziv. Compression of individual sequences via variable rate coding. *IEEE Transaction on Information Theory*, 24:530–536, 1978.

[11] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[12] M. López-Valdés and E. Mayordomo. Dimension is compression. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 676–685. Springer-Verlag, 2005.

[13] E. Mayordomo. Effective fractal dimension in algorithmic information theory. In *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 259–285. Springer-Verlag, 2008.

[14] D. Sculley and C. E. Brodley. Compression and machine learning: A new perspective on feature space vectors. In *Proceedings of the Data Compression Conference (DCC-2006)*, pages 332–341, 2006.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 49-60
www.stacs-conf.org

QUANTUM SEARCH WITH VARIABLE TIMES

ANDRIS AMBAINIS [1]

[1] Department of Computer Science, University of Latvia
Raina bulv. 19, Riga, LV-1586, Latvia
E-mail address: andris.ambainis@lu.lv

Abstract. Since Grover's seminal work, quantum search has been studied in great detail. In the usual search problem, we have a collection of n items $x_1, \ldots, x_n$ and we would like to find $i : x_i = 1$. We consider a new variant of this problem in which evaluating x_i for different i may take a different number of time steps.

Let t_i be the number of time steps required to evaluate x_i. If the numbers t_i are known in advance, we give an algorithm that solves the problem in $O(\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2})$ steps. This is optimal, as we also show a matching lower bound. The case, when t_i are not known in advance, can be solved with a polylogarithmic overhead. We also give an application of our new search algorithm to computing read-once functions.

1. Introduction

Grover's quantum search algorithm [12] is one of two most important quantum algorithms. It allows to search a collection of n items in $O(\sqrt{n})$ quantum steps. This gives a quadratic speedup over the exhaustive search for a variety of search problems [3].

An implicit assumption is that any two items can be examined in the same number of time steps. This is not necessarily true when Grover's algorithm is applied to a specific search problem. It might be the case that some possible solutions to the search problem can be checked faster than others.

Let t_i be the number of time steps required to check the i^{th} solution. Classically, searching for an item $i : x_i = 1$ requires time $\Theta(t_1 + \ldots + t_n)$. A naive application of Grover's search would use $O(\sqrt{n})$ steps, with the maximum possible query time $t_{max} = \max_i t_i$ in each step. This gives a $O(\sqrt{n} t_{max})$ time quantum algorithm.

In this paper, we give a better quantum algorithm. We consider two settings:

(1) The times t_i are known in advance and can be used to design the algorithm;
(2) The times t_i are not known in advance. The algorithm learns t_i only if it runs the computation for checking the i^{th} item for t_i (or more) steps.

For the first setting, we give a quantum algorithm that searches in time $O(\sqrt{T})$ where $T = t_1^2 + \ldots + t_n^2$. For the second, more general setting, we give an $O(\sqrt{T}\log^2 T \log^2 \log T)$ time quantum algorithm. We show a lower bound of $\Omega(\sqrt{T})$ for the first and, hence, also the second setting.

To illustrate the usefulness of our search algorithm, we show an application to computing read-once Boolean functions. A Boolean formula (consisting of AND, OR and NOT operations) $f(x_1, \ldots, x_N)$ is read-once if each of the variables $x_1, \ldots, x_N$ appears at most once in f. We show that any read-once Boolean formula of depth d can be computed using $O(\sqrt{N}\log^{d-1} N)$ queries. The resulting algorithm is weaker than the recent breakthrough work of [4, 11] but is also much simpler than the algorithms in [4, 11].

This is the first paper to construct quantum algorithms for a model in which queries to different x_i take different time. A similar model, however, has been studied in the context of quantum lower bounds by Høyer et al. [14].

Some of the proofs are omitted due to the space constraints. A full version of the paper is available as arXiv preprint `quant-ph/0609168.`

2. Model

We would like to model the situation when the variable x_i is computed by an algorithm A_i which is initialized in the state $|0\rangle$ and, after t_i steps, outputs the final state $|x_i\rangle|\psi_i\rangle$ for some unknown $|\psi_i\rangle$. (For simplicity, we assume that A_i always outputs the correct x_i.) In the first $t_i - 1$ steps, A_i can be in arbitrary intermediate states.

Our goal is to find $i : x_i = 1$. (We sometimes refer to $i : x_i = 1$ as *marked items* and $i : x_i = 0$ as *unmarked*.) Our procedure A can run the algorithms A_i, for some number of steps t, with A_i outputting x_i if $t_i \leq t$ or "the computation is not complete" if $t_i > t$. The computational cost is the amount of time that is spent running algorithms A_i. Any transformations that do not involve A_i are free. This is a generalization of the usual quantum query model.

For completeness, we include a more formal definition of our model in the appendix A. Our algorithms, however, can be understood with just the informal description in the previous two paragraphs.

Known vs. unknown times. We consider two variants of this model. In the "known times" model, the times $t_1, \ldots, t_n$ are known in advance and can be used to design the algorithm. In the "unknown times" model, $t_1, \ldots, t_n$ are unknown to the designer of the algorithm.

3. Methods and subroutines

3.1. Amplitude amplification

Amplitude amplification [8] is a generalization of Grover's quantum search algorithm. Let

$$\sin\alpha|1\rangle|\psi_1\rangle + \cos\alpha|0\rangle|\psi_0\rangle \tag{3.1}$$

be the final state of a quantum algorithm A that outputs 1 with probability $\sin^2\alpha = \delta$. We would like to increase the probability of the algorithm outputting 1. Brassard et al. [8]

showed that, by repeating A and A^{-1} $2m+1$ times, it is possible to generate the final state

$$\sin(2m+1)\alpha|1\rangle|\psi_1\rangle + \cos(2m+1)\alpha|0\rangle|\psi_0\rangle. \tag{3.2}$$

In particular, taking $m = O(\frac{1}{\sqrt{\delta}})$ achieves a constant probability of answer 1.

We use a result by Aaronson and Ambainis [1] who gave a tighter analysis of the same algorithm:

Lemma 3.1. [1] *Let A be a quantum algorithm that outputs a correct answer and a witness with probability*[1] *$\delta \leq \epsilon$ where ϵ is known. Furthermore, let*

$$m \leq \frac{\pi}{4 \arcsin\sqrt{\epsilon}} - \frac{1}{2}. \tag{3.3}$$

Then, there is an algorithm A' which uses $2m+1$ calls to A and A^{-1} and outputs a correct answer and a witness with probability

$$\delta_{new} \geq \left(1 - \frac{(2m+1)^2}{3}\delta\right)(2m+1)^2\delta. \tag{3.4}$$

The distinction between this lemma and the standard amplitude amplification is as follows. The standard amplitude amplification increases the probability from δ to $\Omega(1)$ in $2m+1 = O(\frac{1}{\sqrt{\delta}})$ repetitions. In other words, $2m+1$ repetitions increase the success probability $\Omega((2m+1)^2)$ times. Lemma 3.1 achieves an increase of almost $(2m+1)^2$ times, without the big-Ω factor. This is useful if we have an algorithm with k levels of amplitude amplification nested one inside another. Then, with the usual amplitude amplification, a big-Ω constant of c would result in a c^k factor in the running time. Using Lemma 3.1 avoids that.

We also need another fact about amplitude amplification.

Claim 3.2. *Let δ and δ' be such that $\delta \leq \epsilon$ and $\delta' \leq \epsilon$ and let m satisfy the constraint (3.3). Let $p(\delta)$ be the success probability obtained by applying the procedure of Lemma 3.1 to an algorithm with success probability δ. If $\delta' \leq \delta \leq c\delta'$ for $c \geq 1$, then $p(\delta') \leq p(\delta) \leq cp(\delta')$.*

Proof. Omitted. ∎

3.2. Amplitude estimation

The second result that we use is a version of quantum amplitude estimation.

Theorem 3.3. [8] *There is a procedure* **Est-Amp**(A, M) *which, given a quantum algorithm A and a number M, outputs an estimate $\tilde{\epsilon}$ of the probability ϵ that A outputs 1 and, with probability at least $\frac{8}{\pi^2}$, we have*

$$|\epsilon - \tilde{\epsilon}| \leq 2\pi\frac{\sqrt{\max(\epsilon(1-\epsilon), \tilde{\epsilon}(1-\tilde{\epsilon}))}}{M} + \frac{\pi^2}{M^2}.$$

The algorithm uses M evaluations of A.

We are interested in a slightly different type of error bound. We would like to have $|\epsilon - \tilde{\epsilon}| \leq c\tilde{\epsilon}$ for some small $c > 0$.

[1] [1] requires the probability to be exactly ϵ but the proof works without changes if the probability is less than the given ϵ.

Theorem 3.4. *There is a procedure* **Estimate**(A, c, p, k) *which, given a constant* c, $0 < c \leq 1$ *and a quantum algorithm* A *(with the promise that the probability* ϵ *that the algorithm* A *outputs 1 is either 0 or at least a given value* p*) outputs an estimate* $\tilde{\epsilon}$ *of the probability* ϵ *such that, with probability at least* $1 - \frac{1}{2^k}$*, we have*

(i) $|\epsilon - \tilde{\epsilon}| < c\tilde{\epsilon}$ *if* $\epsilon \geq p$*;*
(ii) $\tilde{\epsilon} = 0$ *if* $\epsilon = 0$*.*

The procedure **Estimate**(A, c, p, k) *uses the expected number of*

$$\Theta\left(k\left(1 + \log\log\frac{1}{p}\right)\sqrt{\frac{1}{\max(\epsilon, p)}}\right)$$

evaluations of A*.*

Proof. Omitted. ∎

4. Search algorithm: known running times

Theorem 4.1. *A collection of* n *items with times* $t_1, \ldots, t_n$ *can be searched in time*

$$O\left(\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}\right).$$

Proof. The basic idea is to subdivide the items into groups so that all items in one group have similar times t_i (e.g. $\frac{t_{max}}{2} \leq t_i \leq t_{max}$ for some t_{max}). We can perform the standard Grover search in a group in time $s = O(\sqrt{l}t_{max})$ where l is the size of the group. We then observe that

$$s^2 = O(lt_{max}^2) = O\left(\sum_i t_i^2\right),$$

with the summation over all items i in the same group. By summing over all groups, we get

$$\sum_j s_j^2 = O\left(\sum_{i=1}^{n} t_i^2\right),$$

where j on the left ranges over all groups. Let k be the number of the groups that we have. If we have a search algorithm that searches k items in time

$$O\left(\sqrt{s_1^2 + \ldots + s_k^2}\right),$$

we can then substitute the algorithms for searching the k groups instead of the k items and obtain a search algorithm for n items that runs in time

$$O\left(\sqrt{t_1^2 + \ldots + t_n^2}\right).$$

We then design a search algorithm for k items in a similar way.

The simplest implementation of this strategy gives an algorithm with $\log^* n$ levels of recursion and running time

$$O\left(c^{\log^* n}\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}\right),$$

due to the reduction from n items to k items losing a constant factor every time it is used. The $c^{\log^* n}$ factor can be avoided, by a more sophisticated implementation of the same idea, which we describe below.

We first restrict to the case when there is exactly one marked item. The general case can be reduced to this case with a constant factor overhead, by running the algorithm on all n elements, a random set of $\frac{n}{2}$, a random set of $\frac{n}{4}$, etc. As shown in [1], there is a constant probability that at least one of those sets contains exactly one marked item. The expected running time increases by at most a constant factor, because of the following lemma.

Lemma 4.2. *Let S be a uniformly random set of $\frac{n}{2^j}$ elements of $\{1, 2, \ldots, n\}$. Then,*

$$E\left[\sqrt{\sum_{i\in S} t_i^2}\right] \leq \frac{1}{2^{j/2}}\sqrt{\sum_{i\in\{1,\ldots,n\}} t_i^2}.$$

Proof. By concavity of the square root function,

$$E\left[\sqrt{\sum_{i\in S} t_i^2}\right] \leq \sqrt{E\left[\sum_{i\in S} t_i^2\right]} = \frac{1}{2^{j/2}}\sqrt{\sum_{i\in\{1,\ldots,n\}} t_i^2}.$$

■

Therefore, the reduction from the general case to one marked item case increases the bound on the number of queries by a factor of at most

$$1 + \frac{1}{2^{1/2}} + \frac{1}{2} + \ldots < \frac{1}{1-\frac{1}{\sqrt{2}}}.$$

Second, we introduce a generalization of the problem in which the algorithm A_i for the marked i returns the correct answer with a probability at least p_i, instead of a certainty. More formally,

- if $x_i = 0$, the final state of the algorithm A_i is of the form $|0\rangle|\psi_0\rangle$.
- if $x_i = 1$, the final state of the algorithm A_i is of the form $\alpha|1\rangle|\psi_1\rangle + \sqrt{1-\alpha^2}|0\rangle|\psi_0\rangle$, where $p_i \leq |\alpha|^2 \leq d \cdot p_i$, for some constant $d > 1$.

The probabilities $p_1, \ldots, p_n$ and the constant d are known to us when we design the algorithm, just as the times $t_1, \ldots, t_n$. (Knowing both the success probability and the running time may look quite artificial. However, we only use the "known success probability" model to design an algorithm for the case when all A_i return the correct answer with certainty.)

We claim that, in this case, we can search in time

$$O\left(\sqrt{\frac{t_1^2}{p_1} + \frac{t_2^2}{p_2} + \ldots + \frac{t_n^2}{p_n}}\right).$$

Our main theorem now follows as the particular case $p_1 = \ldots = p_n = 1$. The main part of our proof is

Lemma 4.3. *There exists $k = O(\log^3 n \log\log n)$ with the following property. Assume that there is a search algorithm for k items with some fixed $d > 1$ that works in time at most*

$$C\sqrt{\frac{s_1^2}{q_1} + \frac{s_2^2}{q_2} + \ldots + \frac{s_k^2}{q_k}}.$$

for any given times $s_1, \ldots, s_k$ and probabilities $q_1, \ldots, q_k$. Then, there exists a search algorithm for n items with $d' = \left(1 - O\left(\frac{1}{\log n}\right)\right) d$ instead of d that works in time at most

$$C\left(1 + O\left(\frac{1}{\log n}\right)\right)\sqrt{\frac{t_1^2}{p_1} + \frac{t_2^2}{p_2} + \ldots + \frac{t_n^2}{p_n}}$$

for any given times $t_1, \ldots, t_n$ and probabilities $p_1, \ldots, p_n$.

Proof. Omitted. ∎

To obtain Theorem 4.1, we repeatedly apply Lemma 4.3 until the number of items becomes less than some constant n_0. That happens after $O(\log^* n)$ applications of Lemma 4.3.

Let $t_1, \ldots, t_n$ and p_1, ..., p_n be the times and probabilities for the final $n \leq n_0$ items. After that, we just amplify the success probability of every item to $\Omega(1)$ (which increases each $\frac{t_i^2}{p_i}$ by at most a constant factor, as discussed in the proof of Lemma 4.3). We then search n items in time $O(\sqrt{n}\max_i t_i)$, using the amplitude amplification, with $\max_i t_i$ steps for evaluating any of the items i. Since $p_i = \Omega(1)$ and $n \leq n_0$ where n_0 is a constant, we have

$$\sqrt{n}\max t_i = O(\max t_i) = O\left(\sqrt{t_1^2 + \ldots + t_n^2}\right) = O\left(\sqrt{\frac{t_1^2}{p_1} + \ldots + \frac{t_n^2}{p_n}}\right).$$

$O(\log^* n)$ applications of Lemma 4.3 increase the time by a factor of at most $(1+O(\frac{1}{\log n}))^{\log^* n} = 1 + o(1)$. ∎

5. Application: read-once functions

A Boolean function $f(x_1, \ldots, x_N)$ that depends on all variables $x_1, \ldots, x_N$ is read-once if it has a Boolean formula (consisting of ANDs, ORs and NOTs) in which every variable appears exactly once. A read-once function can be represented by a tree in which every leaf contains x_i or NOT x_i and every internal vertex contains AND or OR.

Barnum and Saks [5] have shown that, for any read-once f, $\Omega(\sqrt{N})$ queries are necessary to compute f in the quantum query model. Hoyer, Mosca and de Wolf [13] have constructed a $O(\sqrt{N})$ query quantum algorithm for balanced AND-OR trees of constant depth (improving over an earlier $O(\sqrt{N}\log^{d-1} N)$ query algorithm by [10]). In a very recent breakthrough work, [11, 4] showed how to evaluate any AND-OR tree of depth d in $O(\sqrt{Nd})$ queries.

A simple application of our result from the previous section gives a quantum algorithm for evaluating depth-d AND-OR trees. The algorithm is weaker than the one in [11, 4] but is also much simpler.

Theorem 5.1. *Any read-once function $f(x_1, \ldots, x_N)$ of depth d can be computed by a quantum algorithm that uses $O(\sqrt{N}\log^{d-1} N)$ queries.*

Proof. We use induction. If f is represented by a depth-d tree with OR at the root, we express

$$f(x_1, \ldots, x_N) = \vee_{i=1}^{n} f_i(x_{t_1+\ldots+t_{i-1}+1}, \ldots, x_{t_1+\ldots+t_i}).$$

(1) Set $j = 1$. Define B_1 as the algorithm that just outputs 1 and a uniformly random $i \in \{1, \ldots, n\}$.
(2) Repeat:
  (a) Use the algorithm B_j to generate $k = 2\log(D(j+1))$ samples $i_1, \ldots, i_k$ of uniformly random elements $i \in S_j$. Run 2^{j+1} steps of the query procedure on each of $i_1, \ldots, i_k$. If $x_i = 1$ for one of samples, output i and stop.
  (b) Let B'_{j+1} be an algorithm that runs B_j once and, if the output bit is 1, takes the output index i and runs 2^{j+1} steps of the checking procedure on i. If the result is $x_i = 0$, B'_j outputs 0. Otherwise, it outputs 1 and the same index i.
  (c) Let $p = \textbf{Estimate}(B'_{j+1}, c, \frac{1}{N}, 2\log(D(j+1)))$. If $p = 0$, output "no $i : x_i = 0$".
  (d) If $p \geq \frac{1}{9\log n}$, let B_{j+1} be B'_{j+1}.
  (e) If $p < \frac{1}{9\log n}$, let B_{j+1} be the algorithm obtained by amplifying B'_{j+1} $2m+1$ times, where m is the smallest number for which $\frac{1}{9\log n} \leq (2m+1)^2 p \leq \frac{1}{\log n}$. (Such choice of m always exists, as described in the proof of Lemma 4.3.)
  (f) Let $j = j + 1$.

Algorithm 1: Search algorithm for unknown $t_1, \ldots, t_n$

By inductive assumption, we construct algorithms computing the functions f_i in $O(\sqrt{t_i}\log^{d-2} N)$ queries. We then combine them into a quantum algorithm computing f by applying Theorem 4.1.

A more detailed proof is given in the arXiv version of the paper. ∎

6. Search algorithm: unknown running times

In some applications, it may be the case that the times t_i are not known in advance. We can also solve this case, with a polylogarithmic overhead.

Theorem 6.1. *Let $\epsilon > 0$. There is an algorithm that searches collection of n items with unknown times $t_1, \ldots, t_n$ and, with probability at least $1 - \epsilon$, stops after*

$$O\left(T \log^2 T \log^2 \log T\right)$$

steps, where $T = \sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}$.

Proof. Again, we assume that there is exactly one marked item. (The reduction from the general case to the one marked item case is similar to one in the proof of Theorem 4.1.)

Let S_t be the set of items such that $x_i = 1$ or $t_i \geq 2^t$ and let $n_t = |S_t|$. Our main procedure, algorithm 1, defines a sequence of algorithms B_1, ..., B_l. The algorithm B_j, with some success probability, outputs a bit 1 and, conditional on output bit 1, it also outputs a uniformly random index $i \in S_j$. To avoid the problem with accumulating constant factors (described after Lemma 3.1), we make the success probability of B_j slightly less than 1.

Lemma 6.2. *Assume that the constant D in steps 2a and 2c satisfies $D \leq \frac{\pi}{\sqrt{3\epsilon}}$. Then, with probability $1 - \epsilon$, the following conditions are satisfied:*

(a) *Estimates p are accurate within an multiplicative factor of $(1 + c)$;*
(b) *If B_j is defined, then $t_i > 2^{j-1}$ for at least $\frac{n_{j-1}}{2}$ values $i \in \{1, \ldots, n\}$.*

Proof. (a) The probability of error for **Estimate** is at most $\frac{1}{D^2(j+1)^2}$. By summing over all j, the probability of error for some j is at most

$$\frac{1}{D^2}\sum_{i=1}^{\infty}\frac{1}{i^2} = \frac{1}{D^2}\frac{\pi^2}{6},$$

which can be made less than $\frac{\epsilon}{2}$ by choosing $D \leq \frac{\pi}{\sqrt{3\epsilon}}$.

(b) By definition, S_{j-1} is the set of all i with the property that either $x_i = 1$ or $t_i > 2^{j-1}$. Let S be the set of i with $x_i = 1$ and $t_i \leq 2^{j-1}$. If $|S| \leq \frac{1}{2}n_{j-1}$, (c) is true. Otherwise, the probability that each i_j generated in step 2a does not belong to S is less than $\frac{1}{2}$. If one of them belongs to S, algorithm 1 stops without defining B_j. The probability that this does not happen (i.e., all i_j do not belong to S) is less than $(\frac{1}{2})^k = \frac{1}{D^2(j+1)^2}$. We can make this probability arbitrarily small similarly to part (a). ∎

For the rest of the proof, we assume that both conditions of Lemma 6.2 are true. Under this assumption, we bound the running time of algorithm 1. The first step is to bound the running time of the algorithms B_j.

Lemma 6.3. *The running time of B_j is*

$$O\left(j\sqrt{\log n}\sqrt{\frac{t_1^2+t_2^2+\ldots+t_n^2}{n_j}}\right).$$

Proof. Omitted. ∎

We now bound the overall running time. To generate a sample from S_j, one needs $O(\sqrt{\log n})$ invocations of B_j (because the success probability of B_j is of the order $\Omega(\frac{1}{\log n})$). Therefore, we need $O(\sqrt{\log n}\log j)$ invocations to generate $O(\log j)$ samples in step 2a. By Lemma 6.3, that can be done in time

$$O\left(j\log j\log n\sqrt{\frac{t_1^2+t_2^2+\ldots+t_n^2}{n_j}}\right).$$

For each of those samples, we run the checking procedure with 2^{j+1} steps. That takes at most twice the time required by B_j (because B_j includes the checking procedure with 2^j steps). Therefore, the time for the 2^{j+1} checking procedure is of the same order or less than the time to generate the samples.

Second, the success probability estimated in step 2c is of order $\frac{p_j n_{j+1}}{n_j} = \Omega(\frac{n_{j+1}}{n_j \log n})$. By Theorem 3.4, it can be estimated with

$$O\left(\log j\log\log n\sqrt{\frac{n_j\log n}{n_{j+1}}}\right)$$

invocations of B_j, each of which runs in time described by Lemma 6.3.

Thus, the overall number of steps in one loop of algorithm 1 is of order at most

$$\sqrt{t_1^2+t_2^2+\ldots+t_n^2}\left(\frac{j\log j\log n}{\sqrt{n_j}}+\frac{j\log j\log n\log\log n}{\sqrt{n_{j+1}}}\right).$$

Since $n_j \geq 1$ and $n_{j+1} \geq 1$, this is of order

$$O\left(\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2} j \log j \log n \log\log n\right).$$

Let t_{max} be the maximum of $t_1, \ldots, t_n$. Then, the maximum value of j is at most $\lceil \log(t_{max}+1) \rceil$. Therefore, the number of steps used by the algorithm 1 is

$$O\left(\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2} \log n \log\log n \log t_{max} \log\log t_{max}\right).$$

The theorem now follows from $n \leq \sqrt{T}$ and $t_{max} \leq T$, where $T = t_1^2 + t_2^2 + \ldots + t_n^2$. ∎

7. Search lower bound

Theorem 7.1. *For any positive integers $t_1, \ldots, t_n$, searching a collection of n items that can be checked in times $t_1, \ldots, t_n$ requires time $c\sqrt{t_1^2 + t_2^2 + \ldots + t_n^2}$, for some constant $c > 0$.*

Proof. Let t'_i be the minimum positive integer such that $t_i \leq \lceil \frac{\pi}{4}\sqrt{t'_i} \rceil + 1$ (with $t'_i = 1$ if there is no positive integer satisfying this inequality). We consider searching $m = t'_1 + \ldots + t'_n$ elements $x_1, \ldots, x_m \in \{0,1\}$ in the standard model (where every query takes 1 step), with the promise that there is either 0 or 1 element $j : x_j = 1$. By lower bound on quantum search, $c'\sqrt{m}$ queries are required to distinguish between the case when there are 0 elements $j : x_j = 1$ and the case when there is 1 element $j : x_j = 1$, for some constant c'.

We subdivide the inputs $x_1, \ldots, x_m$ into n groups S_1, ..., S_n, with $t'_1, \ldots, t'_n$ elements, respectively. Let $y_i = 1$ if there exists $j \in S_i$ with $x_j = 1$. Since there is either 0 or 1 element $j : x_j = 1$, we know that there is either 0 or 1 element $i : y_i = 1$. We have

Lemma 7.2. *There is an algorithm that implements the transformation $|i\rangle \rightarrow |i\rangle|y_i\rangle|\psi_i\rangle$ for some states $|\psi_i\rangle$, using t_i queries.*

Proof. Omitted.

∎

Let A be a search algorithm for search among n items that require times $t_1, \ldots, t_n$ and let t' be the number of steps used by A. Then, we can substitute the algorithm of Lemma 7.2 instead of the queries y_i. Then, we obtain an algorithm A' that, given $x_1, \ldots, x_n$, asks t' queries and distinguishes whether there is exactly 1 item $i : y_i = 1$ (and, hence, 1 item $j : x_j = 1$) or there is no items $i : y_i = 0$ (and, hence, no items $j : x_j = 1$). Hence,

$$t' \geq c'\sqrt{m} = c'\sqrt{t'_1 + \ldots + t'_n}.$$

We now bound t'_i in terms of t_i. By definition of t'_i, we have

$$t_i \leq \left\lceil \frac{\pi}{4}\sqrt{t'_i} \right\rceil + 1 \leq \frac{\pi}{4}\sqrt{t'_i} + 2.$$

This means that $t'_i \geq \frac{16}{\pi^2}(t_i - 2)^2$. If $t_i \geq 3$, then $t_i - 2 \geq \frac{t_i}{3}$ and $t'_i \geq \frac{16}{9\pi^2}t_i^2$. If $t_i < 3$, then $t'_1 \geq 1 \geq \frac{16}{9\pi^2}t_i^2$. Therefore,

$$t' \geq c'\sqrt{t'_1 + \ldots + t'_n} \geq c'\sqrt{\frac{16}{9\pi^2}(t_1^2 + \ldots + t_n^2)} = \frac{4c'}{3\pi}\sqrt{t_1^2 + \ldots + t_n^2}.$$

This means that the theorem is true, with $c = \frac{4c'}{3\pi}$. ∎

8. Conclusion

In this paper, we gave a quantum algorithm for the generalization of Grover's search in which checking different items requires different times. Our algorithm is optimal for the case when times t_i are known in advance and nearly optimal (within a polylogarithmic factor) for the general case. We also gave an application of our algorithm to computing read-once Boolean functions. It is likely that our algorithms will find other applications.

While we have mostly resolved the complexity of search in this setting, the complexity of other problems have not been studied at all. Of particular interest are problems which are frequently used as a subroutines in other quantum algorithms (for such problems, there is a higher chance that the variable-time query version will be useful). Besides the usual quantum search, the two most common quantum subroutines are quantum counting [9] and k-item search (a version of search in which one has to find k different i for which $x_i = 1$). Element distinctness [2, 6] has also been used as a subroutine, to design quantum algorithms for the triangle problem [16] and verifying matrix identities [7, 15].

Acknowledgements

I would like to thank Robert Špalek and Ronald de Wolf for the discussion that lead to this paper and several anonymous referees for their useful comments. Most of this work done at University of Waterloo, supported by NSERC, ARO, MITACS, ARO and IQC University Professorship.

Currently, my research is supported by University of Latvia Research Grant Y2-ZP01-100.

References

[1] S. Aaronson, A. Ambainis, Quantum search of spatial regions. *Theory of Computing*, 1:47-79, 2005. Also quant-ph/0303041.

[2] A. Ambainis. Quantum walk algorithm for element distinctness. *Proceedings of FOCS'04*, pp. 22-31. Also quant-ph/0311001.

[3] A. Ambainis. Quantum search algorithms. *SIGACT News*, 35 (2004):22-35. Also quant-ph/0504012.

[4] A. Ambainis, A. Childs, B. Reichardt, R. Spalek, S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *Proceedings of FOCS'07*, to appear.

[5] H. Barnum, M. Saks, A lower bound on the quantum complexity of read once functions. *Journal of Computer and System Sciences*, 69:244-258, 2004.

[6] H. Buhrman, C. Durr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, R. de Wolf. Quantum algorithms for element distinctness. *SIAM Journal on Computing*, 34(6): 1324-1330, 2005. Also quant-ph/0007016.

[7] H. Buhrman, R. Špalek: Quantum verification of matrix products. *Proceedings of SODA'06*, pp. 880-889. Also quant-ph/0409035.

[8] G. Brassard, P. Høyer, M. Mosca, A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information Science*, AMS Contemporary Mathematics Series, 305:53-74, 2002. Also quant-ph/0005055.

[9] G. Brassard, P. Høyer, A. Tapp. Quantum counting. *Proceedings of ICALP'98*, pp. 820-831, quant-ph/9805082.

[10] H. Buhrman, R. Cleve, A. Wigderson, Quantum vs. classical communication and computation. *Proceedings of STOC'98*, pages 63-68, quant-ph/9702040.

[11] E. Farhi, J. Goldstone, S. Gutman, A Quantum Algorithm for the Hamiltonian NAND Tree. quant-ph/0702144.

[12] L. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of STOC'96*, pp. 212-219.

[13] P. Høyer, M. Mosca, and R. de Wolf. Quantum search on bounded-error inputs. *Proceedings of ICALP'03*, Lecture Notes in Computer Science, 2719:291-299. Also quant-ph/0304052
[14] P. Høyer, T. Lee, R. Špalek. Tight adversary bounds for composite functions, quant-ph/0509067.
[15] F. Magniez, A. Nayak. Quantum complexity of testing group commutativity. *Algorithmica*, 48(3): 221-232, 2007. Also ICALP'05 and quant-ph/0506265.
[16] F. Magniez, M. Santha, M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2): 413-424, 2007. Also SODA'05 and quant-ph/0310134.

Appendix A. Formal definition of our model

To define our model formally, let $A_i^{(j)}$ be the j^{th} step of A_i. Then,

$$A_i = A_i^{(t_i)} A_i^{(t_i-1)} \dots A_i^{(1)}.$$

We define $A_i^{(t)} = I$ for $t > t_i$. We regard the state space of A_i as consisting of two registers, one of which stores the answer ($c \in \{0,1,2\}$, with 2 representing a computation that has not been completed) and the other register, x, stores any other information.

The state space of a search algorithm is spanned by basis states of the form $|i,t,t_r,c,x,z\rangle$ where $i \in \{1,\dots,n\}$, $t,t_r \in \{0,1,\dots,T\}$ (with T being the number of the query steps in the algorithm), $c \in \{0,1,2\}$ and x and z range over arbitrary finite sets. i represents the index being queried, t represents the number of the time step in which the query for x_i started and t_r is the number of time steps for which A will run the query algorithm A_i. c is the output register of A_i and x holds intermediate data of A_i. Both of those registers should be initialized to $|0\rangle$ at the beginning of every computation of a new x_i. z contains any data that is not a part of the current query.

We define a quantum query algorithm A as a tuple $(U_0,\dots,U_T)$ of unitary transformations that do not depend on $x_1,\dots,x_n$. The actual sequence of transformations that is applied is

$$U_0, Q_1, U_1, Q_2, \dots, U_{T-1}, Q_T, U_T,$$

where Q_j are queries which are defined below. This sequence of transformations is applied to a fixed starting state $|\psi_{start}\rangle$, which consists of basis states $|i,0,0,c,x,z\rangle$.

Queries Q_j are defined in a following way. If $j \le t + t_r$, we apply $A_i^{(j-t)}$ to $|c\rangle$ and $|x\rangle$ registers. Otherwise, we apply I. We call the resulting sequence of queries Q_1, Q_2, ... *generated* by transformations A_i^j. We call Q_1, Q_2 a *valid* sequence of queries corresponding to $x_1,\dots,x_n$ if it is generated by A_i^j satisfying the following constraints:

(1) For $t < t_i$, $A_i^t A_i^{t-1} \dots A_i^1 |0\rangle$ is of the form $|2\rangle|\psi\rangle$ for some $|\psi\rangle$.
(2) For $t = t_i$, $A_i^t A_i^{t-1} \dots A_i^1 |0\rangle$ is of the form $|x_i\rangle|\psi\rangle$ for some $|\psi\rangle$.

U_j can be arbitrary transformations that do not depend on $x_1,\dots,x_n$.

An algorithm $(U_0,\dots,U_T)$ with the starting state $|\psi_{start}\rangle$ computes a function $f(x_1,\dots,x_n)$ if, for every $x_1,\dots,x_n \in \{0,1\}$ and every valid query sequence Q_1, ..., Q_T corresponding to $x_1,\dots,x_n$, the probability of obtaining $f(x_1,\dots,x_n)$ when measuring the first qubit of

$$U_T Q_T U_{T-1} \dots U_1 Q_T U_0 |\psi_{start}\rangle$$

is at least 2/3.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 61-72
www.stacs-conf.org

STRUCTURAL ASPECTS OF TILINGS

ALEXIS BALLIER, BRUNO DURAND, AND EMMANUEL JEANDEL

Laboratoire d'informatique fondamentale de Marseille (LIF) Aix-Marseille Université, CNRS
39 rue Joliot-Curie, 13 453 Marseille Cedex 13, France
E-mail address: alexis.ballier@lif.univ-mrs.fr
E-mail address: bruno.durand@lif.univ-mrs.fr
E-mail address: emmanuel.jeandel@lif.univ-mrs.fr

Abstract. In this paper, we study the structure of the set of tilings produced by any given tile-set. For better understanding this structure, we address the set of finite patterns that each tiling contains.

This set of patterns can be analyzed in two different contexts: the first one is combinatorial and the other topological. These two approaches have independent merits and, once combined, provide somehow surprising results.

The particular case where the set of produced tilings is countable is deeply investigated while we prove that the uncountable case may have a completely different structure.

We introduce a pattern preorder and also make use of Cantor-Bendixson rank. Our first main result is that a tile-set that produces only periodic tilings produces only a finite number of them. Our second main result exhibits a tiling with exactly one vector of periodicity in the countable case.

1. Introduction

Tilings are basic models for geometric phenomena of computation: local constraints they formalize have been of broad interest in the community since they capture geometric aspects of computation [15, 1, 9, 13, 6]. This phenomenon was discovered in the sixties when tiling problems happened to be crucial in logic: more specifically, interest shown in tilings drastically increased when Berger proved the undecidability of the so-called domino problem [1] (see also [8] and the well known book [2] for logical aspects). Later, tilings were basic tools for complexity theory (see the nice review of Peter van Emde Boas [16] and some of Leonid Levin's paper such as [12]).

Because of growing interest for this very simple model, several research tracks were aimed directly on tilings: some people tried to generate the most complex tilings with the most simple constraints (see [15, 9, 13, 6]), while others were most interested in structural aspects (see [14, 5]).

In this paper we are interested in structural properties of tilings. We choose to focus on finite patterns tilings contain and thus introduce a natural preorder on tilings: a tiling

1998 ACM Subject Classification: G.2.m.
Key words and phrases: tiling, domino, patterns, tiling preorder, tiling structure.

is extracted from another one if all finite patterns that appear in the first one also appear in the latter. We develop this combinatorial notion in Section 2.1. This approach can be expressed in terms of topology (subshifts of finite type) and we shall explain the relations between both these approaches in Section 2.2.

It is important to stress that both these combinatorial and topological approaches have independent merits. Among the results we present, different approaches are indeed used for proofs. More specifically, our first main result (Theorem 3.8) states that if a tile-set produces only periodic tilings then it produces only finitely many of them; despite its apparent simplicity, we did not find any proof of Theorem 3.8 in the literature. Our other main result (Theorem 3.11) which states that in the countable case a tiling with exactly one vector of periodicity exists is proved with a strong help of topology.

Our paper is organized as follows: Section 2 is devoted to definitions (combinatorics, topology) and basic structural remarks. In Section 3 we prove the existence of minimal and maximal elements in tilings enforced by a tile-set. Then we present an analysis in terms of Cantor-Bendixson derivative which provides powerful tools. We study the particular case where tilings are countable and present our main results. We conclude by some open problems.

2. Definitions

2.1. Tilings

We present notations and definitions for tilings since several models are used in literature: Wang tiles, geometric frames of rational coordinates, local constraints... All these models are equivalent for our purposes since we consider very generic properties of them (see [3] for more details and proofs). We focus our study on tilings of the plane although our results hold in higher dimensions.

In our definition of tilings, we first associate a state to each cell of the plane. Then we impose a local constraint on them. More formally, Q is a finite set, called the *set of states*. A *configuration* c consists of cells of the plane with states, thus c is an element of $Q^{\mathbb{Z}^2}$. We denote by $c_{i,j}$ or $c(i,j)$ the state of c at the cell (i,j).

A tiling is a configuration which satisfies a given finite set of finite constraints everywhere. More specifically we express these constraints as a set of allowed patterns: a configuration is a tiling if around any of its cells we can see one of the allowed patterns:

Definition 2.1 (patterns)**.** A *pattern* P is a finite restriction of a configuration i.e., an element of Q^V for some finite domain V of $\mathbb{Z}^2$. A pattern *appears* in a configuration c (resp. in some other pattern P') if it can be found somewhere in c (resp. in P'); i.e., if there exists a vector $t \in \mathbb{Z}^2$ such that $c(x+t) = P(x)$ on the domain of P (resp. if $P'(x+t)$ is defined for $x \in V$ and $P'(x+t) = P(x)$) .

By language extension we say that a pattern is *absent* or *omitted* in a configuration if it does not appear in it.

Definition 2.2 (tile-sets and tilings)**.** A *tile-set* is a tuple $\tau = (Q, \mathcal{P}_\tau)$ where $\mathcal{P}_\tau$ is a finite set of patterns on Q. All the elements of $\mathcal{P}_\tau$ are supposed to be defined on the same domain denoted by V ($\mathcal{P}_\tau \subseteq Q^V$).

A *tiling* by τ is a configuration c equal to one of the patterns on all cells:

$$\forall x \in \mathbb{Z}^2, c|_{V+x} \in \mathcal{P}_\tau$$

We denote by $\mathcal{T}_\tau$ the set of tilings by τ.

Notice that in the definition of one tile-set we can allow patterns of different definition domains provided that there are a finite number of them.

An example of a tile-set defined by its allowed patterns is given in Fig. 1. The produced tilings are given in Fig. 2; the meaning of the edges in the graph will be explained later; tilings are represented modulo shift. In A_i and B_i, i is an integer that represents the size of the white stripe.

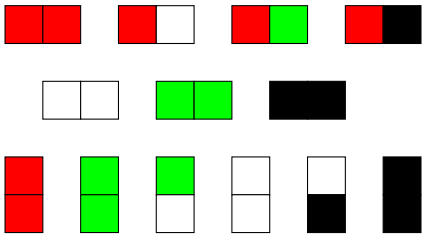
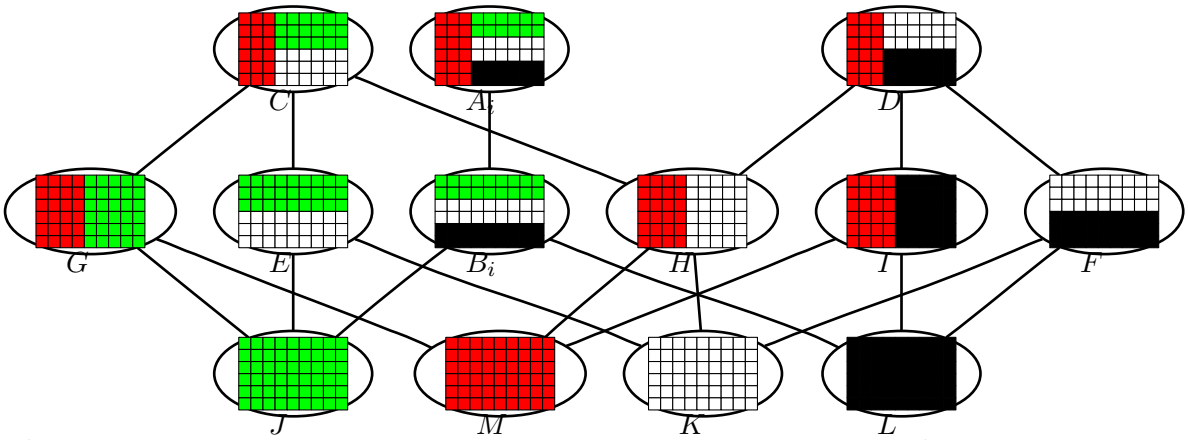


Figure 1: Allowed patterns



An edge represents a relation $Q \prec P$ if P is above Q. Transitivity edges are not depicted. As an example $K \prec E$ and $K \prec C$.

Figure 2: Hasse diagram of the order $\prec$ with the tile-set defined in Fig. 1

Throughout the following, it will be more convenient for us to define tile-sets by the set of their *forbidden patterns*: a tile-set is then given by a finite set $\mathcal{F}_\tau$ of forbidden patterns ($\mathcal{F}_\tau = Q^V \setminus \mathcal{P}_\tau$); a configuration is a tiling if no forbidden pattern appears.

Let us now introduce the following natural preorder, which will play a central role in our paper:

Definition 2.3 (Preorder)**.** Let x, y be two tilings, we say that $x \preceq y$ if any pattern that appears in x also appears in y.

We say that two tilings x, y are equivalent if $x \preceq y$ and $y \preceq x$. We denote this relation by $x \approx y$. In this case, x and y contain the same patterns. The equivalence class of x is denoted by $\langle x \rangle$. We write $x \prec y$ if $x \preceq y$ and $x \not\approx y$.

Some structural properties of tilings can be seen with the help of this preorder. The Hasse diagram in Fig. 2 correspond to the relation $\prec$.

We choose to distinguish two types of tilings: A tiling x is of *type a* if any pattern that appears in x appears infinitely many times; x is of *type b* if there exists a pattern that appears only once in x. Note that any tiling is either of *type a* or of *type b*: suppose that there is a pattern that appears only a finite number of times in x; then the pattern which is the union of those patterns appears only once.

If x is of *type b*, then the only tilings equivalent to x are its shifted: there is a unique way in $\langle x \rangle$ to grow around the unique pattern.

2.2. Topology

In the domain of symbolic dynamics, topology provides both interesting results and is also a nice condensed way to express some combinatorial proofs [10, 7]. The benefit of topology is a little more surprising for tilings since they are essentially static objects. Nevertheless, we can get nice results with topology as will be seen in the sequel.

We see the space of configurations $Q^{\mathbb{Z}^2}$ as a metric space in the following way: the distance between two configurations c and c' is 2^{-i} where i is the minimal offset (for e.g. the euclidean norm) of a point where c and c' differ:

$$d(c, c') = 2^{-\min\{|i|,\ c(i) \neq c'(i)\}}$$

We could also endow Q with the discrete topology and then $Q^{\mathbb{Z}^2}$ with the product topology, thus obtaining the same topology as the one induced by d.

In this topology, a basis of open sets is given through the patterns: for each pattern P, the set $\mathcal{O}_P$ of all configurations c which contains P in their center (i.e., such that c is equal to P on its domain) is an open set, usually called a cylinder. Furthermore cylinders such defined are also closed (their complements are finite unions of $\mathcal{O}_{P'}$ where P' are patterns of same domain different from P). Thus $\mathcal{O}_P$'s are clopen.

Proposition 2.4. *$Q^{\mathbb{Z}^2}$ is a compact perfect metric space (a Cantor space).*

We say that a set of configurations S is shift-invariant if any shifted version of any of its configurations is also in S; i.e., if for every $c \in S$, and every $v \in \mathbb{Z}^2$ the configuration c' defined by $c'(x) = c(x + v)$ is also in S. We denote such a shift by σ_v.

Remark 2.5. Our definition of pattern preorder 2.3 can be reformulated in a topological way : $x \preceq y$ if and only if there exists shifts $(\sigma_i)_{i \in \mathbb{N}}$ such that $\sigma_i(y) \xrightarrow[i \to \infty]{} x$. We say that x can be extracted from y.

For a given configuration x, we define the topological closure of shifted forms of x: $\Gamma(x) = \overline{\{\sigma_v(x),\ v \in \mathbb{Z}^2\}}$ where $\sigma_{i,j}$ represents a shift of vector v.

We see that $x \preceq y$ if and only if $\Gamma(x) \subseteq \Gamma(y)$. Remark that x is minimal for $\prec$ if and only if $\langle x \rangle$ is closed.

As sets of tilings can be defined by a finite number of forbidden patterns, they correspond to *subshifts* of finite type[1]. In the sequel, we sometimes use arbitrary subshifts; they correspond to a set of configurations with a potentially infinite set of forbidden patterns.

3. Main results

3.1. Basic structure

Let us first present a few structural results. First, the existence of minimal classes for $\prec$ is well known.

Theorem 3.1 (minimal elements)**.** *Every set of tilings contains a minimal class for* $\prec$*.*

In the context of tilings, those that belong to minimal classes are often called *quasiperiodic*, while in language theory they are called *uniformly recurrent* or *almost periodic*. Those quasiperiodic configurations admit a nice characterization: any pattern that appears in one of them can be found in any sufficiently large pattern (placed anywhere in the configuration).

For a combinatorial proof of this theorem see [5]. Alternatively, here is a scheme of a topological proof: consider a minimal subshift of $\mathcal{T}_\tau$ (such a subshift exists, see e.g. [14]) then every tiling in this set is in a minimal class.

An intensively studied class of tilings is the set of self-similar tilings. These tilings indeed are minimal elements (quasiperiodic) but one can find other kinds of minimal tilings (e.g. the nice approach of Kari and Culik in [4]).

The existence of maximal classes of tilings is not trivial and we have to prove it:

Theorem 3.2 (maximal elements)**.** *Every set of tilings contains a maximal class for* $\prec$*.*

Proof. Let us prove that any increasing chain has a least upper bound. The theorem is then obtained by Zorn's lemma.

Consider T_i an increasing chain of tiling classes. Consider the set P of all patterns that this chain contains. As the set of all patterns is countable, P is countable too, $P = \{p_i\}_{i\in\mathbb{N}}$.

Now consider two tilings T_k and T_l, any pattern that appears in T_k or T_l appears in $T_{\max(k,l)}$. Thus we can construct a sequence of patterns $(p'_i)_{i\in\mathbb{N}}$ such that p'_i contains all p_j, $j \leq i$ and p'_{i-1}. Note that p'_i is correctly tiled by the considered tile-set.

The sequence of patterns p'_i grows in size. By shift invariance, we can center each p'_i by superimposing an instance of p'_{i-1} found in p'_i over p'_{i-1}. We can conclude that this sequence has a limit and this limit is a tiling that contains all p_i, hence is an upper bound for the chain T_i. ∎

Note that this proof also works when the *set of states* Q and/or the *set of forbidden patterns* $\mathcal{F}_\tau$ are countably infinite (neither compactness nor finiteness is assumed). However it is easy to construct examples where Q is infinite and there does not exist a minimal tiling.

Note that we actually prove that every chain has not only a upper bound, but also a least upper bound. Such a result does not hold for lower bound: We can easily build chains with lower bounds but no greatest lower bound.

[1] *Subshifts* are closed shift-invariant subsets of $Q^{\mathbb{Z}^2}$

3.2. Cantor-Bendixson

In this section we use the topological derivative and define Cantor-Bendixson rank; then we discuss properties of sets of tilings from this viewpoint. Most of the results presented in this section are direct translations of well known results in topology [11].

A configuration c is said to be *isolated* in a set of configurations S if there exists a pattern P (of domain V) such that c is the only configuration in S that contains the pattern P in its center ($\forall x \in V, c(x) = P(x)$). We say that P *isolates* c. This corresponds to the topological notion: a point is isolated if there exists an open set that contains only this point. As an example, in Fig. 3, the tilings A_i are isolated, the pattern isolating an A_i is the boundary between red, white, black and green parts of it.

The topological derivative of a set S is formed by its elements that are not isolated. We denote it by S'.

If S is a set of tilings, or more generally a subshift, we get some more properties. If P isolates a configuration in S then a shifted form of P isolates a shifted form of this configuration. Any configuration of S that contains P is isolated.

As a consequence, if $S = \mathcal{T}_\tau$, then $S' = \mathcal{T}_{\tau'}$ where τ' forbids the set
$\mathcal{F}_\tau \cup \{P | P \text{ isolates some configuration in } \mathcal{T}_\tau\}$.

Note that S' is not always a set of tilings, but remains a subshift. Let us examine the example shown in Fig. 3. S' is S minus the classes A_i. However any set of tilings (subshift of finite type) that contains C, B_i and D also contains A_i. Hence S' is not of finite type in this example.

We define inductively $S^{(\lambda)}$ for any ordinal λ :

- $S^{(0)} = S$
- $S^{(\alpha+1)} = (S^{(\alpha)})'$
- $S^{(\lambda)} = \bigcap_{\alpha<\lambda} S^{(\alpha)}$ when λ is a limit ordinal.

Notice that there exists a countable ordinal λ such that $S^{(\lambda+1)} = S^{(\lambda)}$. Indeed, at each step of the induction, the set of forbidden patterns increases, and there is at most countably many patterns. We call the least such ordinal the *Cantor-Bendixson rank of* S [11].

An element c is *of rank* λ in S if λ is the least ordinal such that $c \notin S^{(\lambda)}$. If no such λ exists, c is of infinite rank. For instance all strictly quasiperiodic configurations (quasiperiodic configurations that are not periodic) are of infinite rank. We write $\rho(x)$ the rank of x.

An example of what Cantor-Bendixson ranks look like is shown in Fig. 3, the first row contains the tilings of rank 1, the second row the ones of rank 2 etc.

Ranked tilings have many interesting properties. First of all, as any $\mathcal{T}_\tau^{(\lambda)}$ is shift-invariant, a tiling has the same rank as its shifted forms.

Note that at each step of the inductive definition, the set of isolated points is at most countable (there are less isolated points than patterns). As a consequence, if all tilings are ranked, $\mathcal{T}_\tau$ is countable, as a countable union (the Cantor-Bendixson rank is countable) of countable sets.

The converse is also true:

Theorem 3.3. *$\mathcal{T}_\tau$ is countable if and only if all tilings are ranked.*

Proof. Let λ be the Cantor-Bendixson rank of $\mathcal{T}_\tau$. $\mathcal{T}_\tau^{(\lambda)} = \mathcal{T}_\tau^{(\lambda+1)}$ is a perfect set (no points are isolated). As a consequence, $\mathcal{T}_\tau^{(\lambda)}$ must be either empty or uncountable (classical

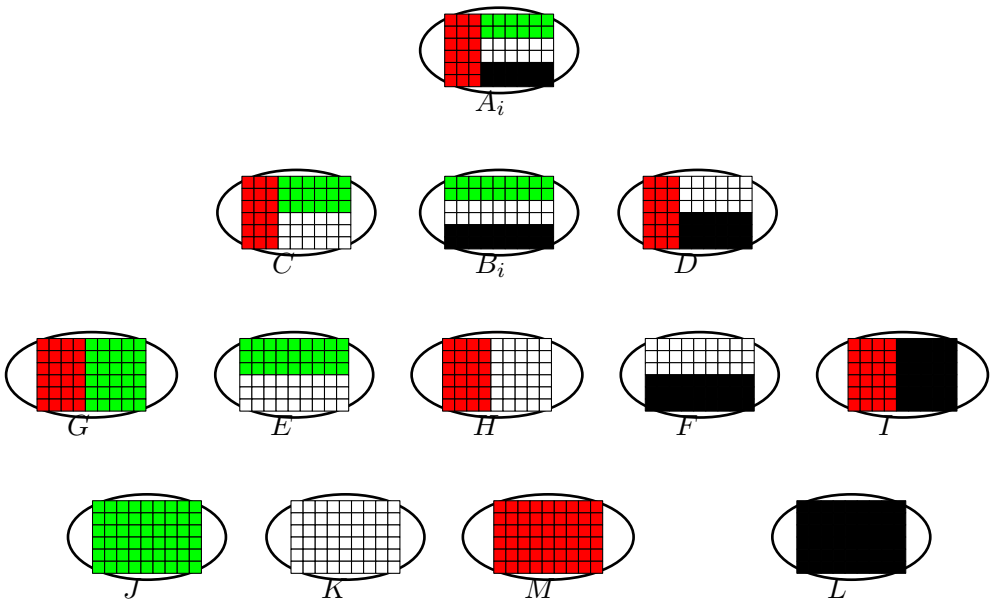


Figure 3: Cantor-Bendixson ranks

application of Baire's Theorem : $\mathcal{T}_\tau^{(\lambda)}$ is compact thus has the Baire property and a non empty perfect set with the Baire property cannot be countable).

As $\mathcal{T}_\tau$ is countable, $\mathcal{T}_\tau^{(\lambda)} = \emptyset$. ∎

Remark 3.4. Strictly quasiperiodic tilings only appear when the number of possible tilings is uncountable [5]. As a consequence, if all tilings are ranked, strictly quasiperiodic tilings do not appear, thus all minimal tilings are periodic. In this case we therefore may expect all tilings to be somehow simple. We'll study this case later in this paper.

As the topology of $Q^{\mathbb{Z}^2}$ has a basis of clopens $\mathcal{C}_P$, $Q^{\mathbb{Z}^2}$ is a 0-dimensional space, thus any subset of $Q^{\mathbb{Z}^2}$ is also 0-dimensional. As any (non empty) perfect 0-dimensional compact metric space is isomorphic to the Cantor Space we obtain:

Theorem 3.5 (Cardinality of tiling spaces)**.** *A set of tilings is either finite, countable or has the cardinality of continuum.*

Note that the proof of this result does not make use of the continuum hypothesis.

We now present the connection between our preorder $\prec$ and the Cantor-Bendixson rank.

Proposition 3.6. *Let x and y be two ranked tilings such that $x \prec y$. Then $\rho(x) > \rho(y)$.*

Proof. By definition of $\prec$, any pattern that appears in x also appears in y. As a consequence, if P isolates x in $S^{(\lambda)}$, then x is the only tiling of $S^{(\lambda)}$ that contains P hence y cannot be in $S^{(\lambda)}$. ∎

Thus tilings of Cantor-Bendixson rank 1 (minimal rank) are maximal tilings for $\prec$. Conversely if all tilings are ranked, tilings of maximal rank exist and are minimal tilings. These tilings are periodic, see remark 3.4.

Another consequence is that if all tilings are ranked, no infinite increasing chain for $\prec$ exists because such chain would induce an infinite decreasing chain of ordinals:

Theorem 3.7. *If $\mathcal{T}_\tau$ is countable, there is no infinite increasing chain for $\prec$.*

3.3. The countable case

In the context of Cantor-Bendixson ranks, the case of countable tilings was revealed as an important particular case. Let us study this case in more details.

If the number of tilings is finite, the situation is easy: any tiling is periodic. Our aim is to prove that in the countable case, there exists a tiling c which has exactly one vector of periodicity (such a tiling is sometimes called weakly periodic in the literature).

We split the proof in three steps :

- There exists a tiling which is not minimal;
- There exists a tiling c which is at level 1, that is such that all tilings less than c are minimal;
- Such a tiling has exactly one vector of periodicity.

The first step is a result of independent interest. To prove the last two steps we use Cantor-Bendixson ranks.

Recall that in our case any minimal tiling is periodic (no strictly quasiperiodic tiling appears in a countable setting [5]). The first step of the proof may thus be reformulated:

Theorem 3.8. *If all tilings produced by a tile-set are periodic, then there are only finitely many of them.*

It is important to note that a compactness argument is not sufficient to prove this theorem, there is no particular reason for a converging sequence of periodic tilings with strictly increasing period to converge towards a non periodic tiling: there indeed exist such sequences with a periodic limit.

Proof. We are in debt to an anonymous referee who simplified our original proof.

Suppose that a tile-set produces infinitely many tilings, but only periodic ones.

As the set of tilings is infinite and compact, one of them is obtained as a limit of the others: There exists a tiling X and a sequence X_i of distinct tilings such that $X_i \to X$.

Now by assumption X is periodic of period p for some p. We may suppose that no X_i has p as a period. Denote by M the pattern which is repeated periodically.

$X_i \to X$ means that X_i contains in its center a square of size $q(i) \times q(i)$ of copies of M, where q is a growing function.

For each i, consider the largest square of X_i consisting only of copies of M. Such a largest square exists, as it is bounded by a period of X_i. Let k be the size of this square. Now, the boundary of this square contains a $p \times p$ pattern which is not M (otherwise this is not the largest square).

By shifting X_i so that this pattern is at the center, we obtain a tiling Y_i which contains a $p \times p$ pattern at the origin which is not M adjacent to a $k/2 \times k/2$ square consisting of copies of M in one of the four quarter planes.

By taking a suitable limit of these Y_i, we will obtain a tiling which contains a $p \times p$ pattern which is not M in its center adjacent to a quarter plane of copies of M.

Such a tiling cannot be periodic. ∎

This proof does not assume that the set of forbidden patterns $\mathcal{F}_\tau$ is finite, therefore it is still valid for any shift-invariant closed subset (subshift) of $Q^{\mathbb{Z}^2}$.

Now we prove stronger results about the Cantor-Bendixson rank of $\mathcal{T}_\tau$. Let α be the Cantor-Bendixson rank of $\mathcal{T}_\tau$. Since $(\mathcal{T}_\tau)^{(\alpha)} = \emptyset$, α cannot be a limit ordinal: Suppose that it is indeed a limit ordinal, therefore $\bigcap_{\beta<\alpha}(\mathcal{T}_\tau)^{(\beta)} = \emptyset$ is an empty intersection of closed sets in $Q^{\mathbb{Z}^2}$ therefore by compactness there exists $\gamma < \alpha$ such that $\bigcap_{\beta<\gamma}(\mathcal{T}_\tau)^{(\beta)} = \emptyset$ and therefore $\mathcal{T}_\tau$ can not have rank α. Hence α is a successor ordinal, $\alpha = \beta + 1$.

However, we can refine this result :

Lemma 3.9. *The rank of $\mathcal{T}_\tau$ cannot be the successor of a limit ordinal.*

Proof. Suppose that $\beta = \cup_{i<\omega}\beta_i$. Since $(\mathcal{T}_\tau)^{(\beta+1)} = \emptyset$, $(\mathcal{T}_\tau)^{(\beta)}$ is finite (otherwise it would have a non-isolated point by compactness), it contains only periodic tilings.

Let p be the least common multiple of the periods of the tilings in $(\mathcal{T}_\tau)^{(\beta)}$. Let M be the set of patterns of size $2p \times 2p$ that do not admit p as a period. Let x_i be an element that is isolated in $(\mathcal{T}_\tau)^{(\beta_i)}$.

As there is only a finite number of p-periodic tilings, we may suppose w.l.o.g. that no x_i admit p as a period.

For any i, there exists a pattern of M that appears in x_i. Let x'_i be the tiling with this pattern at its center. By compactness, one can extract a limit x' of the sequence $(x'_i)_{i\in\mathbb{N}}$, x' is by construction in $\cap_i(\mathcal{T}_\tau)^{(\beta_i)} = \mathcal{T}_\tau^{(\beta)}$. However, x' does not contain a $p-$periodic pattern at its center, that is a contradiction. ∎

We write $\alpha = \lambda + 2$ the rank of $\mathcal{T}_\tau$.

We already proved that there exists a non minimal tiling but this is not sufficient to conclude that there exists a tiling at level 1[2]. However, we achieve this as a corollary of the previous lemma: $(\mathcal{T}_\tau)^{(\lambda)}$ is infinite (otherwise $(\mathcal{T}_\tau)^{(\lambda+1)}$ would be empty) and contains a non periodic tiling by theorem 3.8. This non periodic tiling c is not minimal (otherwise it would be strictly quasiperiodic and then $\mathcal{T}_\tau$ would not be countable). Now c is at level 1 : any tiling less than c is in $(\mathcal{T}_\tau)^{(\lambda+1)}$ therefore periodic (hence minimal).

If a tiling x is of *type a* and is ranked, then it has a vector of periodicity: consider the pattern P that isolates it in the last topological derivative of $\mathcal{T}_\tau$ that it belongs to. Since x is of *type a*, this pattern appears twice in it, therefore there exists a shift σ such that $\sigma(x)$ contains P at its center. $x = \sigma(x)$ because P isolates x.

As any tiling of *type a* has a vector of periodicity, it remains to prove that c is of *type a*:

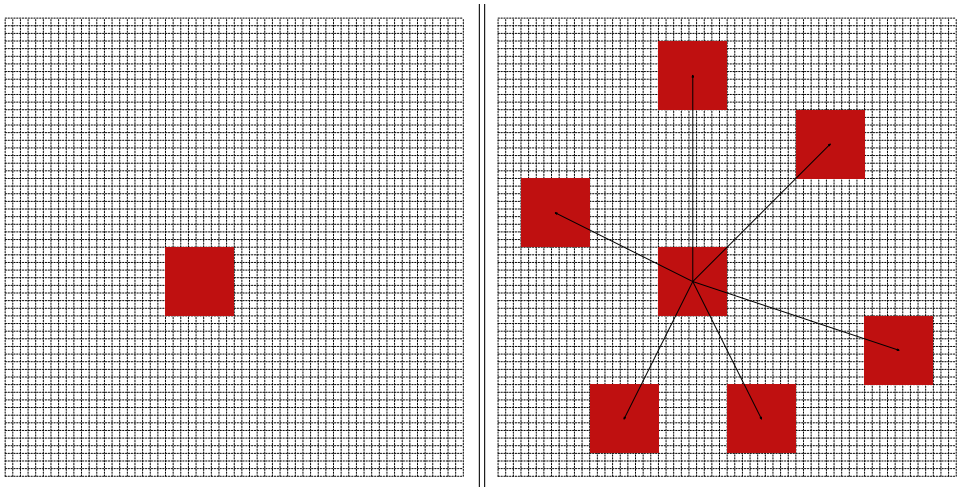
Lemma 3.10. *c is of type a.*

Proof. Suppose the converse : there exists a pattern P that appears only once in c. Considering the union of this pattern P and a pattern that isolates c, we may assume that P isolates c. c has only a finite number of tilings smaller than itself: they lie in $\mathcal{T}_\tau^{(\lambda+1)}$ which is finite, and are all periodic, say of period p. As P isolates c, none of these tilings contain P.

Consider the patterns of size $2p \times 2p$ of T that are not $p-$periodic. If those patterns can appear arbitrary far from P then one can extract a tiling from c (thus smaller than c) that is not $p-$periodic and does not contain P; this is not possible.

[2]We actually can prove that the level 1 exists: There is no infinite decreasing chain whose lower bound is a periodic configuration

Therefore there is a pattern in c that contains P (thus appears only once) and any other part of c is $p-$periodic (one can gather all non $p-$periodic parts of c around P), as depicted in Fig. 4(a).



(a) What we get : c is periodic everywhere but at P

(b) P can appear at many different places since c has periodic patterns

Figure 4: What can happen if c is of *type b*?

This non periodic part could also be inserted at infinitely many different positions in c since the tiling rules are of bounded radius, as depicted in Fig. 4(b). Hence the number of tilings is not countable. ∎

c is of *type a*, c is not periodic, c has a vector of periodicity, therefore our theorem 3.11 holds :

Theorem 3.11. *If τ is a tile-set that produces a countable number of tilings then it produces a tiling with exactly one vector of periodicity.*

4. Open problems

We are interested in proving more precise results for the order $\prec$ for a countable set of tilings: we wonder whether the order $\prec$ has at most finitely many levels, as it is the case in Fig. 2. We know how to construct a tile-set so that the maximal level is any arbitrary integer see e.g. Fig.5 for level 3.

We also intend to prove a similar result for uncountable sets of tilings; the problem is that we are tempted to think that if the set of tilings is uncountable, then a quasiperiodic tiling must appear. However, this is not true: imagine a tile-set that admits a vertical line of white or black cells with red on the left and green on the right. The uncountable part is due to the vertical line that itself contains a quasiperiodic of dimension 1 but not of dimension 2. This tile-set produces tilings that looks like H in Fig. 2, except that the vertical line can have two different colors without any constraint.

A generalization of lemma 3.9 would be to prove that the Cantor-Bendixson rank of a countable set of tilings cannot be infinite; we know how to construct sets of tilings that have

an arbitrary large but finite Cantor-Bendixson rank, but we do not know how to obtain a set of tilings of rank greater than ω.

References

[1] R. Berger. The undecidability of the domino problem. *Memoirs American Mathematical Society*, 66:1966, 1966.

[2] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

[3] Julien Cervelle and Bruno Durand. Tilings: recursivity and regularity. *Theor. Comput. Sci.*, 310(1-3):469–477, 2004.

[4] Karel Culik and Jarkko Kari. On aperiodic sets of Wang tiles. *Lecture Notes in Computer Science*, 1337:153–??, 1997.

[5] Bruno Durand. Tilings and quasiperiodicity. *Theor. Comput. Sci.*, 221(1-2):61–75, 1999.

[6] Bruno Durand, Leonid A. Levin, and Alexander Shen. Complex tilings. In *STOC*, pages 732–739, 2001.

[7] Walter Helbig Gottschalk and Gustav Arnold Hedlund. *Topological Dynamics*. American Mathematical Society, Providence, Rhode Island, 1955.

[8] Y. Gurevich and I. Koriakov. A remark on Berger's paper on the domino problem. *Siberian Journal of Mathematics*, 13:459–463, 1972. (in Russian).

[9] William P. Hanf. Nonrecursive tilings of the plane. i. *J. Symb. Log.*, 39(2):283–285, 1974.

[10] G. A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Mathematical Systems Theory*, 3:320–375, 1969.

[11] Kazimierz Kuratowski. *Topology, Vol. I, 3rd edition*. NY: Academic Press, 1966.

[12] Leonid A. Levin. Average case complete problems. *SIAM J. Comput.*, 15(1):285–286, 1986.

[13] Dale Myers. Nonrecursive tilings of the plane. ii. *J. Symb. Log.*, 39(2):286–294, 1974.

[14] C. Radin and M. Wolff. Space tilings and local isomorphism, 1992.

[15] R. M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12:177–209, 1971.

[16] P. van Embde Boas. Dominoes are forever. Research report 83-04, University of Amsterdam. Department of Mathematics., 1983.

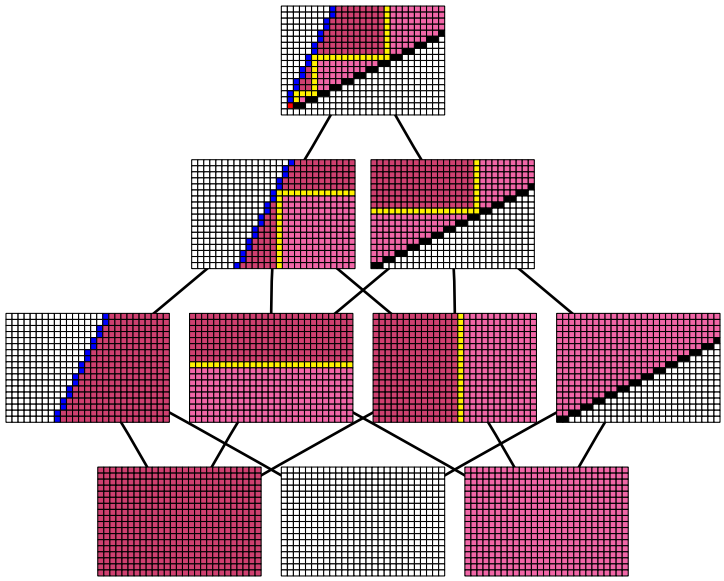


Figure 5: An example of a tile-set that produces countably many tilings and a tiling at level 3

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 73-84
www.stacs-conf.org

LIMIT COMPLEXITIES REVISITED

LAURENT BIENVENU [1], ANDREJ MUCHNIK [2], ALEXANDER SHEN [3], AND NIKOLAY VERESHCHAGIN [4]

[1] Laboratoire d'Informatique Fondamentale
CNRS & Université de Provence,
39 rue Joliot Curie, F-13453 Marseille cedex 13
E-mail address: Laurent.Bienvenu@lif.univ-mrs.fr

[2] Andrej Muchnik (24.02.1958 – 18.03.2007)
worked in the Institute of New Technologies in Education, Moscow

[3] Laboratoire d'Informatique Fondamentale,
Poncelet Laboratory, CNRS, IITP RAS, Moscow
E-mail address: Alexander.Shen@lif.univ-mrs.fr

[4] Moscow State Lomonosov University, Russia
E-mail address: ver@mccme.ru

Abstract. The main goal of this paper is to put some known results in a common perspective and to simplify their proofs.

We start with a simple proof of a result from [7] saying that $\limsup_n C(x|n)$ (here $C(x|n)$ is conditional (plain) Kolmogorov complexity of x when n is known) equals $C^{\mathbf{0}'}(x)$, the plain Kolmogorov complexity with $\mathbf{0}'$-oracle.

Then we use the same argument to prove similar results for prefix complexity (and also improve results of [4] about limit frequencies), a priori probability on binary tree and measure of effectively open sets. As a by-product, we get a criterion of $\mathbf{0}'$ Martin-Löf randomness (called also 2-randomness) proved in [3]: a sequence ω is 2-random if and only if there exists c such that any prefix x of ω is a prefix of some string y such that $C(y) \geqslant |y| - c$. (In the 1960ies this property was suggested in [1] as one of possible randomness definitions; its equivalence to 2-randomness was shown in [3] while proving another 2-randomness criterion (see also [5]): ω is 2-random if and only if $C(x) \geqslant |x| - c$ for some c and infinitely many prefixes x of ω.

Finally, we show that the low-basis theorem can be used to get alternative proofs for these results and to improve the result about effectively open sets; this stronger version implies the 2-randomness criterion mentioned in the previous sentence.

Key words and phrases: Kolmogorov complexity, limit complexities, limit frequencies, 2-randomness, low basis.

1. Plain complexity

By $C(x)$ we mean the plain complexity of a binary string x (the length of the shortest description of x when an optimal description method is fixed, see [2]; no requirements about prefixes). By $C(x|n)$ we mean conditional complexity of x when n is given [2]. Superscript $\mathbf{0}'$ in $C^{\mathbf{0}'}$ means that we consider the relativized (with oracle $\mathbf{0}'$, the universal enumerable set) version of complexity.

The following result was proved in [7]. We provide a simple proof for it.

Theorem 1.1.

$$\limsup_{n\to\infty} C(x|n) = C^{\mathbf{0}'}(x) + O(1).$$

Proof. We start with the easy part. Let $\mathbf{0}_n$ be the (finite) part of the universal enumerable set that appeared after n steps. If $C^{\mathbf{0}'}(x) \leqslant k$, then there exists a description (program) of size at most k that generates x using $\mathbf{0}'$ as an oracle. Only finite part of the oracle can be used, so $\mathbf{0}'$ can be replaced by $\mathbf{0}_n$ for all sufficiently large n, and oracle $\mathbf{0}_n$ can be reconstructed if n is given as a condition. Therefore, $C(x|n) \leqslant k + O(1)$ for all sufficiently large n, and

$$\limsup_{n\to\infty} C(x|n) \leqslant C^{\mathbf{0}'}(x) + O(1).$$

Now fix k and assume that $\limsup C(x|n) < k$. This means that for all sufficiently large n the string x belongs to the set

$$U_n = \{u \mid C(u|n) < k\}.$$

The family U_n is an enumerable family of sets (given n and k, we generate U_n); each of these sets has less than 2^k elements. We need to construct a $\mathbf{0}'$-computable process that given k generates at most 2^k elements, and among them all elements that belong to U_n for all sufficiently large n. (Then strings of length k may be assigned as $\mathbf{0}'$-computable codes of all generated elements.)

To describe this process, consider the following operation: for some u and N add u to all U_n such that $n \geqslant N$. (In other terms, we add a horizontal ray starting from (N, u) to the set $\mathcal{U} = \{(n, u) \mid u \in U_n\}$.) This operation is *acceptable* if all U_n still have less than 2^k elements after it (i.e., if before this operation all U_n such that $n \geqslant N$ either contain u or have less than $2^k - 1$ elements).

For given u and k we can find out using $\mathbf{0}'$-oracle whether this operation is acceptable. Now for all pairs (N, u) (in some computable order) we perform (N, u)-operation if it is acceptable. (The elements added to some U_i remain there and are taken into account when next operations are attempted.) This process is $\mathbf{0}'$-computable since after any finite number of operations the family $\mathcal{U}$ is enumerable (without any oracle) and its enumeration algorithm can be $\mathbf{0}'$-effectively found (uniformly in k).

Therefore the set of all elements u that participate in acceptable operations during this process is uniformly $\mathbf{0}'$-enumerable. This set contains less than 2^k elements (otherwise U_n would become too big for large n). Finally, this set contains all u such that u belongs to the (initial) U_n for all sufficiently large n. Indeed, the operation is always acceptable if all added elements are already present. ∎

The proof has the following structure. We have an enumerable family of sets U_n that have less than 2^k elements. This implies that the set

$$U_\infty = \liminf_{n\to\infty} U_n$$

has less than 2^k elements (the lim inf of a sequence of sets is the set of elements that belong to almost all sets of the sequence). If this set were $\mathbf{0}'$-enumerable, we would be done. However, this may be not the case: the criterion

$$u \in U_\infty \Leftrightarrow \exists N\, (\forall n \geqslant N)\, [u \in U_n]$$

has $\exists\forall$ prefix before an enumerable (not necessarily decidable) relation, that is, one quantifier more than we want (to guarantee that U_∞ is $\mathbf{0}'$-enumerable). However, in our proof we managed to cover U_∞ by a set that is $\mathbf{0}'$-enumerable and still has less than 2^k elements.

2. Prefix complexity and a priori probability

Now we prove similar result for prefix complexity (or, in other terms, for a priori probability). Let us recall the definition. The function $a(x)$ on binary strings (or integers) with non-negative real values is called a *semimeasure* if $\sum_x a(x) \leqslant 1$. The function a is *lower semicomputable* if there exists a computable total function $(x,n) \mapsto a(x,n)$ with rational values such that for every x the sequence $a(x,0), a(x,1), \ldots$ is a non-decreasing sequence that has limit $a(x)$.

There exists a maximal (up to a constant factor) lower semicomputable semimeasure m. The value $m(x)$ is sometimes called the *a priori probability* of x. In the same way we can define *conditional* a priory probability $m(x|n)$ and $\mathbf{0}'$*-relativized* a priori probability $m^{\mathbf{0}'}(x)$.

Theorem 2.1.

$$\liminf_{n\to\infty} m(x|n) = m^{\mathbf{0}'}(x)$$

up to a $\Theta(1)$ *factor.*

(In other terms, two inequalities with $O(1)$ factors hold.)

Proof. If $m^{\mathbf{0}'}(x)$ is greater that some ε, then for some k the increasing sequence $m^{\mathbf{0}'}(x,k)$ that has limit $m^{\mathbf{0}'}(x)$ becomes greater than ε. The computation of $m^{\mathbf{0}'}(x,k)$ uses only finite amount of information about the oracle, thus for all sufficiently large n we have $m^{\mathbf{0}_n}(x) \geqslant m^{\mathbf{0}_n}(x,k) > \varepsilon$. So, similar to the previous theorem, we have

$$\liminf_{n\to\infty} m(x|n) \geqslant \liminf_{n\to\infty} m^{\mathbf{0}_n}(x) \geqslant m^{\mathbf{0}'}(x)$$

up to $O(1)$ factors.

In the other direction the proof is also similar to the previous one. Instead of enumerable finite sets U_n now we have a sequence of (uniformly) lower semicomputable functions $x \mapsto m_n(x) = m(x|n)$. Each of m_n is a semimeasure. We need to construct a lower $\mathbf{0}'$-semicomputable semimeasure m' such that

$$m'(x) \geqslant \liminf_{n\to\infty} m_n(x)$$

Again, the lim inf itself cannot be used as m': though $\sum_x \liminf_n m_n(x) < 1$ if $\sum_x m_n(x) \leqslant 1$ for all n, but, unfortunately, the equivalence

$$r < \liminf_{n\to\infty} a_n \Leftrightarrow (\exists r' > r)(\exists N)\,(\forall n \geqslant N)\,[r' < a_n]$$

has too many quantifier alternations (one more than needed; note that lower semicomputable a_n makes [...] condition enumerable). The similar trick helps. For a triple (r, N, u) consider an *increase operation* that increases all values $m_n(u)$ such that $n \geqslant N$ up to a given rational number r (not changing them if they were greater than or equal to r). This operation is *acceptable* if all m_n remain semimeasures after the increase.

The question whether operation is acceptable is $\mathbf{0}'$-decidable; if it is, we get a new (uniformly) lower semicomputable (without any oracle) sequence of semimeasures and can repeat an attempt to perform an increase operation for some other triple. Doing that for all triples (in some computable ordering), we can then define $m'(u)$ as the upper bound of r for all successful (r, N, u) increase operations (for all N). This gives a $\mathbf{0}'$-lower semicomputable function; it is a semimeasure since we verify the semimeasure inequality for every successful increase attempt; finally, $m'(u) \geqslant \liminf m_n(u)$ since if $m_n(u) \geqslant r$ for all $n \geqslant N$, then (r, N, u)-increase does not change anything and is guaranteed to be acceptable. ∎

The expression $-\log m(x)$ equals the so-called *prefix* complexity $K(x)$ (up to $O(1)$ term; see [2]). The same is true for relativized and conditional versions, an we get the following reformulation of the last theorem:

Theorem 2.2.

$$\limsup_{n\to\infty} K(x|n) = K^{\mathbf{0}'}(x) + O(1).$$

Another corollary improves a result of [4]. For any (partial) function f from $\mathbb{N}$ to $\mathbb{N}$ we define the *limit frequency* of an integer x as

$$q_f(x) = \liminf_{n\to\infty} \frac{\#\{i < n \mid f(i) = x\}}{n}$$

In other words, we look at the fraction of x-terms in $f(0), \ldots, f(n-1)$ (undefined values are also listed) and take lim inf of these frequencies. It is easy to see that for a total computable f the function q_f is a lower $\mathbf{0}'$-semicomputable semimeasure. The argument above proves the following result:

Theorem 2.3. *For any partial computable f the function q_f is upper bounded by a lower $\mathbf{0}'$-semicomputable semimeasure.*

In [4] it is shown that for some total computable f the function q_f is a maximal lower $\mathbf{0}'$-semicomputable semimeasure and therefore $\mathbf{0}'$-relativized a priori probability can be defined as maximal limit frequency for total computable functions. Now we see that the same is true for partial computable functions: allowing them to be partial does not increase the maximal limit frequency.

The similar argument also is applicable to the so-called *a priori complexity* defined as negative logarithm of a maximal lower semicomputable semimeasure on the binary tree (see [8]). This complexity is sometimes denoted as $KA\,(x)$ and we get the following statement:

Theorem 2.4.

$$\limsup_{n\to\infty} KA\,(x|n) = KA^{\,\mathbf{0}'}(x) + O(1).$$

(To prove this we define an increase operation in such a way that it increases not only $a(x)$ but also $a(y)$ for y that are prefixes of x, if necessary. The increase is acceptable if $a(\Lambda)$ still does not exceed 1.)

It would be interesting to find out whether similar results are true for monotone complexity or not (the authors do not know this).

3. Open sets of small measure

We now try to apply the same trick in a slightly different situation, for effectively open sets. The Cantor space Ω is a set of all infinite sequence of zeros and ones. An *interval* Ω_x (for a binary string x) is formed by all sequences that have prefix x. Open sets are unions of intervals. An *effectively open* subset of Ω is an enumerable union of intervals, i.e., the union of intervals Ω_x where x are takes from some enumerable set of strings.

We consider standard (uniform Bernoulli) measure on Ω: the interval Ω_x has measure 2^{-l} where l is the length of x.

A classical theorem of measure theory says: *if $U_0, U_1, U_2, \ldots$ are open sets of measure at most ε, then $\liminf_n U_n$ has measure at most ε, and this implies that for every $\varepsilon' > \varepsilon$ there exists an open set of measure at most ε' that covers $\liminf_n U_n$.*

Indeed,

$$\liminf_{n\to\infty} U_n = \bigcup_N \bigcap_{n\geqslant N} U_n,$$

and the measure of the union of an increasing sequence

$$V_N = \bigcap_{n\geqslant N} U_n,$$

equals the limit of measures of V_N, and all these measures do not exceed ε since $V_N \subset U_N$. It remains to note that for any measurable set X its measure is the infimum of the measures of open sets that cover X.

We now can try to "effectivize" this statement in the same way as we did before. First we started with an (evident) statement: *if U_n are finite sets of at most 2^k elements, then $\liminf_n U_n$ has at most 2^k elements* and proved its effective version: *for a uniformly enumerable family of open sets U_n that have at most 2^k elements, the set $\liminf_n U_n$ is contained in a uniformly $\mathbf{0}'$-enumerable set that has at most 2^k elements.* Then we did similar thing with semimeasures (again, the non-effective version is trivial: it says that if $\sum_x m_n(x) \leqslant 1$ for every n, then $\sum_x \liminf_n m_n(x) \leqslant 1$).

Now the effective version could look like this. *Let $\varepsilon > 0$ be a rational number and let $U_0, U_1, \ldots$ be an enumerable family of effectively open sets of measure at most ε each. Then for every rational $\varepsilon' > \varepsilon$ there exists a $\mathbf{0}'$-effectively open set of measure at most ε' that contains $\liminf_{n\to\infty} U_i = \bigcup_N \bigcap_{n\geqslant N} U_n$.*

However, the authors do not know whether this is always true. The argument that we have used can nevertheless be applied do prove the following weaker version:

Theorem 3.1. *Let $\varepsilon > 0$ be a rational number and let U_n be an enumerable family of effectively open sets of measure at most ε each. Then there exists a uniformly $\mathbf{0}'$-effectively open set of measure at most ε that contains*

$$\bigcup_N \mathrm{Int}\Big(\bigcap_{n\geqslant N} U_n\Big)$$

Here $\mathrm{Int}(X)$ denotes the interior part of X, i.e., the union of all open subsets of X. In this case we do not need ε' (which one could expect since the union of open sets is open).

Proof. Following the same scheme, for every string x and integer N we consider (x, N)-operation that adds Ω_x to all U_n such that $n \geqslant N$. This operation is *acceptable* if measures of all U_n remain at most ε for each n. This can be checked using $\mathbf{0}'$-oracle (if the operation is not acceptable, it becomes known after a finite number of steps).

We attempt to perform this operation (if acceptable) for all pairs in some computable order. The union of all added intervals for all accepted pairs is $\mathbf{0}'$-effectively open. If some sequence belongs to the union of the interior parts, then it is covered by some interval Ω_u that is a subset of U_n for all sufficiently large n. Then some (u, N)-operation is acceptable since it actually does not change anything and therefore Ω_u is a part of an $\mathbf{0}'$-open set that we have constructed. ∎

4. Kolmogorov and 2-randomness

This result has an historically remarkable corollary. When Kolmogorov tried to define randomness in 1960ies, he started with the following approach. A sequence x of length n is "random" if its complexity $C(x)$ (or conditional complexity $C(x|n)$; in fact, these requirements are almost equivalent) is close to n: the *randomness deficiency* $d(x)$ is defined as the difference $|x| - C(x)$ (here $|x|$ stands for the length of x). This sounds reasonable, but if we then define a random sequence as a sequence whose prefixes have deficiencies bounded by a constant, such a sequence does not exist at all: Martin-Löf showed that every infinite sequence has prefixes of arbitrarily large deficiency, and suggested a different definition of randomness using effectively null sets. Later more refined versions of randomness deficiency (using monotone or prefix complexity) appeared that make the criterion of randomness in terms of deficiencies possible. But before that, in 1968, Kolmogorov wrote: "The most natural definition of infinite Bernoulli sequence is the following: x is considered m-Bernoulli type if m is such that all x^i are *initial segments* of the finite m-Bernoulli sequences. Martin-Löf gives another, possibly narrower definition" ([1], p. 663).

Here Kolmogorov speaks about "m-Bernoulli" finite sequence x (this means that $C(x|n, k)$ is greater than $\log \binom{n}{k} - m$ where n is the length of x and k is the number of ones in x). For the case of uniform Bernoulli measure (where $p = q = 1/2$) one would reformulate this definition as follows. Let us define

$$\bar{d}(x) = \inf\{d(y) \mid x \text{ is a prefix of } y\}$$

and require that $\bar{d}(x)$ is bounded for all prefixes of an infinite sequence ω. It is shown by J. Miller in [3] that this definition is equivalent to Martin-Löf randomness relativized to $\mathbf{0}'$ (called also 2-*randomness*):

Theorem 4.1. *A sequence ω is Martin-Löf $\mathbf{0}'$-random if and only if the quantities $\bar{d}(x)$ for all prefixes x of ω are bounded by a (common) constant.*

In turns out that this result (in one direction) easily follows from the previous theorem.

Proof. Assume that $\bar{d}$-deficiencies for prefixes of ω are not bounded. According to Martin-Löf definition, we have to construct for a given c an $\mathbf{0}'$-effectively open set that covers ω and has measure at most 2^{-c}.

Fix some c. For each n consider the set D_n of all sequences u of length n such that $C(u) < n-c$ (i.e., sequences u of length n such that $d(u) > c$). It has at most 2^{n-c} elements. The requirement $\bar{d}(x) > c$ means that every string extension y of x belongs to D_m where m is its length. This implies that Ω_x is contained in every U_m where $m \geqslant |x|$ and U_m is the set of all sequences that have prefixes in D_m (this set has measure at most 2^{-c}). Therefore, in this case the interval Ω_x is a subset of $\bigcap_{m \geqslant |x|} U_m$ and (being open) is a subset of its interior. Then we conclude (using the result proved above) that Ω_x (=every sequence with prefix x) is covered by an $\mathbf{0}'$-effectively open set of measure at most 2^{-c} constructed as explained above. So if some ω has prefixes of arbitrarily large $\bar{d}$-deficiency, then ω is not $\mathbf{0}'$ Martin-Löf random.

Note that this argument works also for conditional complexity (with length as condition) and gives a slightly stronger result.

For the sake of completeness we reproduce (from [3]) the proof of the reverse implication (essentially unchanged). Assume that a sequence ω is covered (for each c) by a $\mathbf{0}'$-computable sequence of intervals $I_0, I_1, \ldots$ of total measure at most 2^{-c}. (We omit c in our notation, but all these constructions depend on c.)

Using the approximations $\mathbf{0}_n$ instead of full $\mathbf{0}'$ and performing at most n steps of computation for each n we get another (now computable) family of intervals $I_{n,0}, I_{n,1}, \ldots$ such that $I_{n,i} = I_i$ for every i and sufficiently large n. We may assume without loss of generality that $I_{n,i}$ either has size at least 2^{-n} (i.e., is determined by a string of length at most n) or equals $\bot$ (a special value that denotes the empty set) since only the limit behavior is prescribed. Moreover, we may also assume that $I_{n,i} = \bot$ for $i > n$ and that the total measure of all $I_{n,0}, I_{n,1}, \ldots$ does not exceed 2^{-c} for every n (by deleting the excessive intervals in this order; the stabilization guarantees that all limit intervals will be eventually let through).

Since $I_{n,i}$ is defined by intervals of size at least 2^{-n}, we get at most 2^{n-c} strings of length n covered by intervals $I_{n,i}$ for given n and all i. This set is decidable (recall that only i not exceeding n are used), therefore each string in this set can be defined (assuming c is known) by a string of length $n-c$, binary representation of its ordinal number in this set. (Note that this string also determines n if c is known.)

Returning to the sequence ω, we note that it is covered by some I_i and therefore is covered by $I_{n,i}$ for this i and all sufficiently large n (after the value is stabilized), say, for all $n \geqslant N$. Let u be a prefix of ω of length N. All continuations of u of any length n are covered by $I_{n,i}$ and have complexity less than $n - c + O(1)$. In fact, this is a conditional complexity with condition c; we get $n - c + 2\log c + O(1)$, so $\bar{d}(u) \geqslant c - 2\log c - O(1)$.

Such a string u can be found for every c, therefore ω has prefixes of arbitrarily large $\bar{d}$-deficiency. ∎

In fact a stronger statement than Theorem 4.1 is proved in [3, 5]; our tools are still too weak to get this statement. However, the low basis theorem helps.

5. The low basis theorem

This is a classical result in recursion theory (see, e.g., [6]). It was used in [5] to prove 2-randomness criterion; analyzing this proof, we get theorems about limit complexities as byproducts. For the sake of completeness we reproduce the statement and the proof of low-basis theorem here; they are quite simple.

Theorem 5.1. *Let $U \subset \Omega$ be an effectively open set that does not coincide with Ω. Then there exists a sequence $\omega \notin U$ which is low, i.e., $\omega' = \mathbf{0}'$*

Here ω' is the jump of ω; the equation $\omega' = \mathbf{0}'$ means that the universal ω-enumerable set is $\mathbf{0}'$-decidable.

Theorem 5.1 says that any effectively closed non-empty set contains a low element. For example, if $P, Q \subset \mathbb{N}$ are enumerable inseparable sets, then the set of all separating sequences is an effectively closed set that does not contain computable sequences. We conclude, therefore, that there exists a non-computable low separating sequence.

Proof. Assume that an oracle machine M and an input x are fixed. The computation of M with oracle ω on x may terminate or not depending on oracle ω. Let us consider the set $T(M, x)$ of all ω such that $M^\omega(x)$ terminates (for fixed machine M and input x). This set is an effectively open set (if termination happens, it happens due to finitely many oracle values). This set together with U may cover the entire Ω; this means that $M^\omega(x)$ *terminates for all* $\omega \notin U$. If it is not the case, we can add $T(M, x)$ to U and get a bigger effectively open set U' that still has non-empty complement such that $M^\omega(x)$ *does not terminate for all* $\omega \in U'$. This operation guarantees (in one of two ways) that termination of the computation $M^\omega(x)$ does not depend on the choice of ω (in the remaining non-empty effectively closed set).

This operation can be performed for all pairs (M, x) sequentially. Note that if $U \cup T(M, x)$ covers the entire Ω, this happens on some finite stage (compactness), so $\mathbf{0}'$ is enough to find out whether it happens or not, and on the next step we have again some effectively open (without any oracle) set. So $\mathbf{0}'$-oracle is enough to say which of the computations $M^\omega(x)$ terminate (as we have said, this does not depend of the choice of ω). Therefore any such ω is low (the universal ω-enumerable set is $\mathbf{0}'$-decidable). And such an ω exists since the intersection of the decreasing sequence of non-empty closed sets is non-empty (compactness). ∎

6. Using the low basis theorem

Let us show how Theorem 1.1 can be proved using the low basis theorem. As we have seen, we have an enumerable family of sets U_n that have at most 2^k elements and need to construct effectively a $\mathbf{0}'$-enumerable set that has at most 2^k elements and contains $U_\infty = \liminf_n U_n$.

If the sets U_n are (uniformly) decidable, then U_∞ is $\mathbf{0}'$-enumerable and we do not need any other set. The low basis theorem allows us to reduce general case to this special one. Let us consider the family of all "upper bounds" for U_n: by an upper bound we mean a sequence V_n of finite sets that contain U_n and still have at most 2^k elements each. The sequence $V_0, V_1, \ldots$ can be encoded as an infinite binary sequence (first we encode V_0, then V_1 etc.; note that each V_i can be encoded by a finite number of bits though this number depends on V_i).

For a binary sequence the property "to be an encoding of an upper bound for U_n" is effectively closed (the restriction $\#V_n < 2^k$ is decidable and the restriction $U_n \subset V_n$ is co-enumerable). Therefore the low basis theorem can be applied. We get an upper bound V that is low. Then $V_\infty = \liminf V_n$ is (uniformly in k) V'-enumerable (as we have said: with V-oracle the family V_n is uniformly decidable), but since V is low, V'-oracle can be replaced by $\mathbf{0}'$-oracle, and we get the desired result.

This proof though being simple looks rather mysterious: we get something almost out of nothing! (As far as we know, this idea in a more advanced context appeared in [5].)

The same trick can be used to prove Theorem 2.1: here "upper bounds" are distributions M_n with rational values and finite support that are greater than $m(x|n)$ but still are semimeasures. (Technical correction: first we have to assume that $m(x|n) = 0$ if x is large, and then we have to weaken the restriction $\sum M_n(x) \leqslant 1$ replacing 1 by, say, 2; this is needed since the values $m(x|n)$ may be irrational.)

Theorem 2.4 can be also proved in this way (upper bounds should be semimeasures on tree with rational values and finite support).

As to Theorem 3.1, here the application of the low basis theorem allows us to get a stronger result than before (though not the most strong version we mentioned as an open question):

Theorem 6.1. *Let $\varepsilon > 0$ be a rational number and let U_n be an uniformly enumerable family of effectively open sets, i.e.,*

$$U_n = \cup\{\Omega_x \mid (n, x) \in U\}$$

for some enumerable set $U \subset \mathbb{N} \times \{0,1\}^*$. Assume that U_n has measure at most ε for every n. Assume also that U_i has "effectively bounded granularity", i.e., all strings x such that $(n, x) \in U$ have length at most $c(n)$ where c is a total computable function. Then for every $\varepsilon' > \varepsilon$ there exists a $\mathbf{0}'$-effectively open set W of measure at most ε' that contains

$$\liminf_{n\to\infty} U_n = \bigcup_N \bigcap_{n\geqslant N} U_n$$

and this construction is uniform.

Proof. First we use the low basis theorem to reduce the general case to the case where U is decidable and for every $(n, x) \in U$ the length of x is exactly $c(n)$.

Indeed, define an "upper bound" as a sequence V of sets V_n where V_n is a set of strings of length $c(n)$ such that U_n is covered by the intervals generated by elements of V_n. Again V can be encoded as an infinite sequence of zeros and ones, and the property "to be an upper bound" is effectively closed. Applying the low basis theorem, we choose a low V and add it is an oracle. Since V' is equivalent to $\mathbf{0}'$, for our purpose we may assume that V is decidable.

Now we have to deal with the decidable case. Let us represent the set U_∞ as a union of the disjoint sets

$$F_0 = \bigcap_i U_i,\ F_1 = \bigcap_{i\geqslant 1} U_i \setminus U_0,\ F_2 = \bigcap_{i\geqslant 2} U_i \setminus U_1, \ldots$$

(for each element x in U_∞ we consider the last U_i that does not contain x). Each of F_i is (in the decidable case) an effectively closed set (recall than U_i is open-closed due to the restriction on $c(i)$). Moreover, the F_i are pairwise disjoint and the family F_i satisfies

$$\liminf_{n\to+\infty} U_n = \bigcup_i F_i$$

and thus

$$\sum_i \mu(F_i) = \mu(\liminf_{n\to+\infty} U_n).$$

The measure of each of F_i is $\mathbf{0}'$-computable, and using $\mathbf{0}'$-oracle we can find a finite set of intervals that covers F_i and has measure

$$\mu(F_i) + (\varepsilon' - \varepsilon)/2^{i+1}$$

Putting all these intervals together, we get the desired set W. So the decidable case (and therefore the general one, thanks to low basis theorem) is completed. ∎

7. Corollary on 2-randomness

Theorem 6.1 can be used to prove 2-randomness criterion from [3, 5]. In fact, this gives exactly the proof from [5]; the only thing we did is structuring the proof in two parts (formulating Theorem 6.1 explicitly and putting it in the context of other results on limits of complexities).

Theorem 7.1 ([3, 5])**.** *A sequence ω is $\mathbf{0}'$ Martin-Löf random if and only if*

$$C(\omega_0\omega_1\ldots\omega_{n-1}) \geqslant n - c$$

for some c and for infinitely many n.

Proof. Let us first understand the relation between this theorem and Theorem 4.1. If

$$C(\omega_0\omega_1\ldots\omega_{n-1}) \geqslant n - c$$

for infinitely many n and given c, then $\bar{d}(x) \leqslant c$ for every prefix x of ω (indeed, one can find the required continuation of x among prefixes of ω). As we know, this guarantees that ω is $\mathbf{0}'$ Martin-Löf random.

It remains to prove that if for all c we have

$$C(\omega_0\omega_1\ldots\omega_{n-1}) < n - c$$

for all sufficiently large n, then ω is not $\mathbf{0}'$-random. Using the same notation as in the proof of Theorem 4.1, we can say that ω has a prefix in D_n and therefore belongs to U_n for all sufficiently large n. We can apply then Theorem 6.1 since U_n is defined using strings of length n (so $c(n) = n$) and cover U_∞ (and therefore ω) by a $\mathbf{0}'$-effectively open set of small measure. Since this can be uniformly done for all c, the sequence ω is not $\mathbf{0}'$-random. ∎

Remark. The results above may be considered as special cases of an effective version of a classical theorem in measure theory: Fatou's lemma. This lemma guarantees that if $\int f_n(x)\, d\mu(x) \leqslant \varepsilon$ for μ-measurable functions $f_0, f_1, f_2, \ldots$, then

$$\int \liminf_{n\to+\infty} f_n(x)\, d\mu(x) \leqslant \varepsilon.$$

The constructive version assumes that f_i are lower semicomputable and satisfy some additional conditions; it says that for every $\varepsilon' > \varepsilon$ there exists a lower $\mathbf{0}'$-semicomputable function φ such that $\liminf f_n(x) \leqslant \varphi(x)$ for every x and $\int \varphi(x) d\mu(x) \leqslant \varepsilon'$.

References

[1] Kolmogorov A.N., Logical Basis for Information Theory and Probability Theory. *IEEE Transactions on Information Theory*, v. IT-14, No. 5, Sept. 1968. (Russian version was published in 1969.)

[2] Li M., Vitányi P., *An Introduction to Kolmogorov Complexity and Its Applications*, Second Edition, Springer, 1997. (638 pp.)

[3] Miller J., Every 2-random real is Kolmogorov random, *Journal of Symbolic Logic*, **69**(2):555–584 (2004).

[4] Muchnik An.A., Lower limits of frequencies in computable sequences and relativized a priori probability, *SIAM Theory Probab. Appl., 1987, vol. 32, p. 513–514.*

[5] Nies A., Stephan F., Terwijn S., Randomness, relativization and Turing degrees, *Journal of Symbolic Logic*, **70**(2):515–535 (2005).

[6] Odifreddi P., *Classical recursion theory*, North-Holland, 1989.

[7] Vereshchagin N. K. Kolmogorov complexity conditional to large integers. *Theoretical Computer Science*, v. 271 (2002), issues 1–2, p. 59–67.

[8] Zvonkin A.K., Levin L. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Math. Surveys*, **25**:6 (1970), p. 83–124.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 85-96
www.stacs-conf.org

TRIMMED MOEBIUS INVERSION AND GRAPHS OF BOUNDED DEGREE

ANDREAS BJÖRKLUND [1], THORE HUSFELDT [1], PETTERI KASKI [2], AND MIKKO KOIVISTO [2]

[1] Lund University, Department of Computer Science, P.O.Box 118, SE-22100 Lund, Sweden
E-mail address: andreas.bjorklund@logipard.com, thore.husfeldt@cs.lu.se

[2] Helsinki Institute for Information Technology HIIT, University of Helsinki, Department of Computer Science, P.O.Box 68, FI-00014 University of Helsinki, Finland
E-mail address: {petteri.kaski,mikko.koivisto}@cs.helsinki.fi

Abstract. We study ways to expedite Yates's algorithm for computing the zeta and Moebius transforms of a function defined on the subset lattice. We develop a trimmed variant of Moebius inversion that proceeds point by point, finishing the calculation at a subset before considering its supersets. For an n-element universe U and a family $\mathcal{F}$ of its subsets, trimmed Moebius inversion allows us to compute the number of packings, coverings, and partitions of U with k sets from $\mathcal{F}$ in time within a polynomial factor (in n) of the number of supersets of the members of $\mathcal{F}$.

Relying on an intersection theorem of Chung *et al.* (1986) to bound the sizes of set families, we apply these ideas to well-studied combinatorial optimisation problems on graphs of maximum degree Δ. In particular, we show how to compute the Domatic Number in time within a polynomial factor of $(2^{\Delta+1}-2)^{n/(\Delta+1)}$ and the Chromatic Number in time within a polynomial factor of $(2^{\Delta+1}-\Delta-1)^{n/(\Delta+1)}$. For any constant Δ, these bounds are $O\big((2-\epsilon)^n\big)$ for $\epsilon > 0$ independent of the number of vertices n.

1. Introduction

Yates's algorithm from 1937 is a kind of fast Fourier transform that computes for a function $f : \{0,1\}^n \to \mathbf{R}$ and another function $\upsilon : \{0,1\} \times \{0,1\} \to \mathbf{R}$ the values

$$\widehat{f}(x_1,\ldots,x_n) \;=\; \sum_{y_1,\ldots,y_n \in \{0,1\}} \upsilon(x_1,y_1)\cdots\upsilon(x_n,y_n) f(y_1,\ldots,y_n)\,. \tag{1.1}$$

simultaneously for all $X = (x_1,\ldots,x_n) \in \{0,1\}^n$ using only $O(2^n n)$ operations, instead of the obvious $O(4^n n)$. The algorithm is textbook material in many sciences. Yet, though it appears in Knuth [13, §3.2], it has received little attention in combinatorial optimisation.

Recently, the authors [3, 4] used Yates's algorithm in combination with Moebius inversion to give algorithms for a number of canonical combinatorial optimisation problems such as Chromatic Number and Domatic Number in n-vertex graphs, and n-terminal Minimum Steiner Tree, in running times within a polynomial factor of 2^n.

This research was supported in part by the Academy of Finland, Grants 117499 (P.K.) and 109101 (M.K.).

From the way it is normally stated, Yates's algorithm seems to face an inherent 2^n lower bound, up to a polynomial factor, and it also seems to be oblivious to the structural properties of the transform it computes.

The motivation of the present investigation is to expedite the running time of Yates's algorithm for certain structures so as to get running times with a dominating factor of the form $(2-\epsilon)^n$. From the perspective of running times alone, our improvements are modest at best, but apart from providing evidence that the aesthetically appealing 2^n bound from [4] can be beaten, the combinatorial framework we present seems to be new and may present a fruitful direction for exact exponential time algorithms.

1.1. Results

In a graph $G = (V, E)$, a set $D \subseteq V$ of vertices is *dominating* if every vertex not in D has at least one neighbour in D. The *domatic number* of G is the largest k for which V can be partitioned in to k dominating sets. We show how to compute the domatic number of an n-vertex graph with maximum degree Δ in time

$$O^*\big((2^{\Delta+1} - 2)^{n/(\Delta+1)}\big)\,;$$

the O^* notation suppresses factors that are polynomial in n. For constant Δ, this bound is always better than 2^n, though not by much:

Δ	3	4	5	6	7	8	$\cdots$
$(2^{\Delta+1} - 2)^{1/(\Delta+1)}$	1.9344	1.9744	1.9895	1.9956	1.9981	1.9992	$\cdots$

The *chromatic number* of a graph is the minimum k for which the vertex set can be covered with k independent sets; a set $I \subseteq V$ is *independent* if no two vertices in I are neighbours. We show how to compute the chromatic number of an n-vertex graph with maximum degree Δ in time

$$O^*\big((2^{\Delta+1} - \Delta - 1)^{n/(\Delta+1)}\big)\,.$$

This is slightly faster than for Domatic Number:

Δ	3	4	5	6	7	8	$\cdots$
$(2^{\Delta+1} - \Delta - 1)^{1/(\Delta+1)}$	1.8613	1.9332	1.9675	1.9840	1.9921	1.9961	$\cdots$

One notes that even for moderate Δ, the improvement over 2^n is minute. Moreover, the colouring results for $\Delta \leq 5$ are not even the best known: by Brooks's Theorem [5], the chromatic number of a connected graph is bounded by its maximum degree unless the graph is complete or an odd cycle, both of which are easily recognised. It remains to decide if the chromatic number is 3, 4, or 5, and with algorithms from the literature, 3- and 4-colourability can be decided in time $O(1.3289^n)$ [1] and $O(1.7504^n)$ [6], respectively. However, this approach does stop at $\Delta = 5$, since we know no $o(2^n)$ algorithm for 5-colourability. Other approaches for colouring low-degree graphs are known via pathwidth: given a path decomposition of width w the k-colourability can be decided in time $k^w n^{O(1)}$ [11]; for 6-regular graphs one can find a decomposition with $w \leq n(23+\epsilon)/45$ for any $\epsilon > 0$ and sufficiently large n [11], and for graphs with m edges one can find $w \leq m/5.769 + O(\log n)$ [12]. However, even these pathwidth based bounds fall short when $k \geq 5$—we are not aware of any previous $o(2^n)$ algorithm.

For the general case, it took 30 years and many papers to improve the constant in the bound for Chromatic Number from 2.4423 [14] via 2.4151 [9], 2.4023 [6], 2.3236 [2],

Figure 1: Trimmed evaluation. Originally, Yates's algorithm considers the entire subset lattice (left). We trim the evalation from below by considering only the supersets of 'interesting' points (middle), and from above by abandoning computation when we reach certain points (right).

to 2 [4], and a similar (if less glacial) story can be told for the Domatic Number. None of these approaches was sensitive to the density of the graph. Moreover, what interests us here is not so much the size of the constant, but the fact that it is less than 2, dispelling the tempting hypothesis that 2^n should be a 'difficult to beat' bound for computing the Chromatic Number for sparse graphs. In §4 we present some tailor-made variants for which the running time improvement from applying the ideas of the present paper are more striking.

Chromatic Number and Domatic Number are special cases of set partition problems, where the objective is to partition an n-element set U (here, the vertices of a graph) with members of a given family $\mathcal{F}$ of its subsets (here, the independent or dominating sets of the graph). In full generality, we show how to compute the covering, packing, and partition numbers of $(U, \mathcal{F})$ in time within a polynomial factor of

$$\left|\{T \subseteq U : \text{there exists an } S \in \mathcal{F} \text{ such that } S \subseteq T\}\right|, \tag{1.2}$$

the number of supersets of the members of $\mathcal{F}$. In the worst case, this bound is not better than 2^n, and the combinatorial challenge in applying the present ideas is to find good bounds on the above expression.

1.2. Techniques

The main technical contribution in this paper, sketched in Figure 1, is that Yates's algorithm can, for certain natural choices of $\upsilon : \{0,1\} \times \{0,1\} \to \mathbf{R}$, be trimmed by considering in a bottom-up fashion only those $X \in \{0,1\}^n$ that we are actually interested in, for example those X for which $f(X) \neq 0$ and their supersets. (We will understand X as a subset of $\{1, \ldots, n\}$ whenever this is convenient.) Among the transforms that are amenable to trimming are the zeta and Moebius transforms on the subset lattice.

We use the trimmed algorithms for zeta and Moebius transforms to expedite Moebius inversion, a generalisation of the principle of inclusion–exclusion, which allows us to compute the cover, packing, and partition numbers. The fact that these numbers can be computed via Moebius inversion was already used in [2, 3, 4], and those parts of the present paper contain little that is new, except for a somewhat more explicit and streamlined presentation in the framework of partial order theory.

The fact that we can evaluate both the zeta and Moebius transforms pointwise in such a way that we are done with X before we proceed to Y for every $Y \supset X$ also enables us to further trim computations from what is outlined above. For instance, if we seek a minimum set partition of sets from a family $\mathcal{F}$ of subsets of U, then it suffices to find the minimum partition of all X such that $U \setminus X = S$ for some $S \in \mathcal{F}$. In particular, we need not consider how many sets it takes to partition X for X's large enough for $U \setminus X$ not to contain any set from $\mathcal{F}$.

The main combinatorial contribution in this paper is that if $\mathcal{F}$ is the family of maximal independent sets, or the family of dominating sets in a graph, then we show how to bound (1.2) in terms of the maximum degree Δ using an intersection theorem of Chung *et al.* [8] that goes back to Shearer's Entropy Lemma. For this we merely need to observe that

the intersection of $\mathcal{F}$ and the closed neighbourhoods of the input graph excludes certain configurations.

In summary, via (1.2) the task of bounding the running time for (say) Domatic Number reduces to a combinatorial statement about the intersections of certain families of sets.

Notation. Yates's algorithm operates on the lattice of subsets of an n-element universe U, and we find it convenient to work with notation established in partial order theory.

For a family $\mathcal{F}$ of subsets of U, let $\min \mathcal{F}$ (respectively, $\max \mathcal{F}$) denote the family of minimal (respectively, maximal) elements of $\mathcal{F}$ with respect to subset inclusion. The *upper closure* (sometimes called *up-set* or *filter*) of $\mathcal{F}$ is defined as

$$\uparrow\mathcal{F} = \{\, T \subseteq U : \text{there exists an } S \in \mathcal{F} \text{ such that } S \subseteq T \,\}.$$

For a function f defined on subsets of U, the *support* of f is defined as

$$\operatorname{supp}(f) = \{\, X \subseteq U : f(X) \neq 0 \}.$$

For a graph G, we let $\mathcal{D}$ denote the family of *dominating sets* of G and $\mathcal{I}$ the family of *independent sets* of G. Also, for a subset $W \subseteq V$ of vertices, we let $G[W]$ denote the subgraph induced by W. For a proposition P, we use Iverson's bracket notation $[P]$ to mean 1 if P is true and 0 otherwise.

2. Trimmed Moebius Inversion

For a family $\mathcal{F}$ of sets from $\{0,1\}^n$ and a set $X \in \{0,1\}^n$ we will consider k-tuples $(S_1, \ldots, S_k)$ with $S_i \in \mathcal{F}$ and $S_i \subseteq X$. Such a tuple is *disjoint* if $S_{i_1} \cap S_{i_2} = \emptyset$ for all $1 \leq i_1 < i_2 \leq k$, and *covering* if $S_1 \cup \cdots \cup S_k = X$. From these concepts we define for fixed k

(1) the *cover number* $c(X)$, viz. the number of covering tuples,
(2) the *packing number* $p(X)$, viz. the number of disjoint tuples,
(3) the *partition number* or *disjoint cover number* $d(X)$, viz. the number of tuples that are both disjoint and covering.

In this section we show how to compute these numbers in time $|\uparrow\mathcal{F}|n^{O(1)}$, rather than $2^n n^{O(1)}$ as in [3, 4]. The algorithms are concise but somewhat involved, and we choose to present them here starting with an explanation of Yates's algorithm. Thus, the first two subsections are primarily expository and aim to establish the new ingredients in our algorithms.

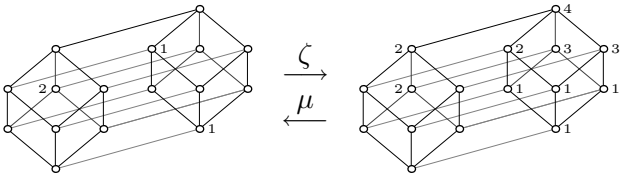
At the heart of our algorithms lie two transforms of functions $f : \{0,1\}^n \to \mathbf{R}$ on the subset lattice. The *zeta* transform $f\zeta$ is defined for all $X \in \{0,1\}^n$ by

$$(f\zeta)(X) = \sum_{Y \subseteq X} f(Y). \tag{2.1}$$

(The notation $f\zeta$ can be read either as a formal operator or as a product of the 2^n-dimensional vector f and the matrix ζ with entries $\zeta_{YX} = [Y \subseteq X]$.) The *Moebius* transform $f\mu$ is defined for all $X \in \{0,1\}^n$ by

$$(f\mu)(X) = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} f(Y). \tag{2.2}$$

These transforms are each other's inverse in the sense that $f = f\zeta\mu = f\mu\zeta$, a fundamental combinatorial principle called *Moebius inversion*. We can (just barely) draw an example in four dimensions for a function f given by $f(\{4\}) = f(\{1,2,4\}) = 1$, $f(\{1,3\}) = 2$ and $f(X) = 0$ otherwise:



Another example that we will use later is the connection between the packing number and the disjoint cover number,

$$p = d\zeta\,, \tag{2.3}$$

which is easy to verify: By definition,

$$(d\zeta)(X) \;=\; \sum_{Y\subseteq X} d(Y)\,.$$

Every disjoint k-tuple $(S_1,\ldots,S_k)$ with $S_1\cup\cdots\cup S_k \subseteq X$ appears once on the right hand side, namely for $Y = S_1\cup\cdots\cup S_k$, so this expression equals the packing number $p(X)$.

2.1. Yates's algorithm

Yates's algorithm [17] expects the transform in the form of a function $\upsilon : \{0,1\}\times\{0,1\}\to\mathbf{R}$ and computes the transformed values

$$\widehat{f}(X) \;=\; \sum_{Y\in\{0,1\}^n} \upsilon(x_1,y_1)\cdots\upsilon(x_n,y_n) f(Y)\,. \tag{2.4}$$

simultaneously for all $X\in\{0,1\}^n$. Here, we let $(x_1,\ldots,x_n)$ and $(y_1,\ldots,y_n)$ denote the binary representations (or, 'incidence vectors') of X and Y, so $x_j = [j\in X]$ and $y_j = [j\in Y]$. To obtain (2.1) set $\upsilon(x,y) = [y\le x]$ and to obtain (2.2) set $\upsilon(x,y) = [y\le x](-1)^{x-y}$.

The direct evaluation of (2.4) would take 2^n evaluations of f for each X, for a total of $O(2^n 2^n n) = O(4^n n)$ operations. The zeta and Moebius transforms depend only on $Y\subseteq X$, so they would require only $\sum_X 2^{|X|} = \sum_{0\le i\le n}\binom{n}{i}2^i = 3^n$ evaluations. Yates's algorithm is faster still and computes the general form in $O(2^n n)$ operations:

Algorithm Y. (*Yates's algorithm.*) Computes $\widehat{f}(X)$ defined in (2.4) for all $X\in\{0,1\}^n$ given $f(Y)$ for all $Y\in\{0,1\}^n$ and $\upsilon(x,y)$ for all $x,y\in\{0,1\}$.

Y1: For each $X\in\{0,1\}^n$, set $g_0(X) = f(X)$.

Y2: For each $j = 1,\ldots,n$ and $X\in\{0,1\}^n$, set

$$g_j(X) = \upsilon([j\in X],0)g_{j-1}(X\setminus\{j\}) + \upsilon([j\in X],1)g_{j-1}(X\cup\{j\})\,.$$

Y3: Output g_n.

The intuition is to compute $\widehat{f}(X)$ 'coordinate-wise' by fixing fewer and fewer bits of X in the sense that, for $j = 1,\ldots,n$,

$$g_j(X) \;=\; \sum_{y_1,\ldots,y_j\in\{0,1\}} \upsilon(x_1,y_1)\cdots\upsilon(x_j,y_j) f(y_1,\ldots,y_j,x_{j+1},\ldots,x_n)\,. \tag{2.5}$$

Indeed, the correctness proof is a straightforward verification (by induction) of the above expression.

2.2. Trimmed pointwise evaluation

To set the stage for our present contributions, observe that both the zeta and Moebius transforms 'grow upwards' in the subset lattice in the sense that $\mathrm{supp}(f\zeta), \mathrm{supp}(f\mu) \subseteq \uparrow\mathrm{supp}(f)$. Thus, in evaluating the two transforms, one ought to be able to trim off redundant parts of the lattice and work only with lattice points in $\uparrow\mathrm{supp}(f)$.

We would naturally like trimmed evaluation to occur in $O(|\uparrow\mathrm{supp}(f)|n)$ operations, in the spirit of Algorithm Y. However, to obtain the values at X in Step Y2 of Algorithm Y, at first sight it appears that we must both 'look up' (at $X \cup \{j\}$) and 'look down' (at $X \setminus \{j\}$). Fortunately, it suffices to only 'look down'. Indeed, for the zeta transform, setting $\upsilon(x, y) = [y \leq x]$ and simplifying Step Y2 yields

$$g_j(X) = [j \in X]g_{j-1}(X \setminus \{j\}) + g_{j-1}(X)\,. \tag{2.6}$$

For the Moebius transform, setting $\upsilon(x, y) = [y \leq x](-1)^{x-y}$ and simplifying yields

$$g_j(X) = -[j \in X]g_{j-1}(X \setminus \{j\}) + g_{j-1}(X)\,. \tag{2.7}$$

Furthermore, it is not necessary to look 'too far' down: for both transforms it is immediate from (2.5) that

$$g_j(X) = 0 \text{ holds for all } X \notin \uparrow\mathrm{supp}(f) \text{ and } j = 0, \ldots, n\,. \tag{2.8}$$

In what follows we tacitly employ (2.8) to limit the scope of (2.6) and (2.7) to $\uparrow\mathrm{supp}(f)$.

The next observation is that the lattice points in $\uparrow\mathrm{supp}(f)$ can be evaluated in order of their rank, using sets $\mathcal{L}(r)$ containing the points of rank r. Initially, the sets $\mathcal{L}(r)$ contain only $\mathrm{supp}(f)$, but we add elements from $\uparrow\mathrm{supp}(f)$ as we go along. These observations result in the following algorithm for evaluating the zeta transform; the algorithm for evaluating the Moebius transform is obtained by replacing (2.6) in Step Z3 with (2.7).

Algorithm Z. (*Trimmed pointwise fast zeta transform.*) Computes the nonzero part of $f\zeta$ given the nonzero part of f. The algorithm maintains $n+1$ families $\mathcal{L}(0), \ldots, \mathcal{L}(n)$ of subsets $X \in \{0,1\}^n$; $\mathcal{L}(r)$ contains only sets of size r. We compute auxiliary values $g_j(X)$ for all $1 \leq j \leq n$ and $X \in \uparrow\mathrm{supp}(f)$; it holds that $g_n(X) = (f\zeta)(X)$.

Z1: For each $X \in \mathrm{supp}(f)$, insert X into $\mathcal{L}(|X|)$. Set the current rank $r = 0$.

Z2: Select any $X \in \mathcal{L}(r)$ and remove it from $\mathcal{L}(r)$.

Z3: Set $g_0(X) = f(X)$. For each $j = 1, \ldots, n$, set

$$g_j(X) = [j \in X]g_{j-1}(X \setminus \{j\}) + g_{j-1}(X)\,.$$

[At this point $g_n(X) = (f\zeta)(X)$.]

Z4: If $g_n(X) \neq 0$, then output X and $g_n(X)$.

Z5: For each $j \notin X$, insert $X \cup \{j\}$ into $\mathcal{L}(r+1)$.

Z6: If $\mathcal{L}(r)$ is empty then increment $r \leq n$ until $\mathcal{L}(r)$ is nonempty; terminate if $r = n$ and $\mathcal{L}(n)$ is empty.

Z7: Go to Z2.

Observe that the evaluation at X is complete once Step Z3 terminates, which enables further trimming of the lattice 'from above' in case the values at lattice points with higher rank are not required.

By symmetry, the present ideas work just as well for transforms that 'grow downwards', in which case one needs to 'look up'. However, they do not work for transforms that grow in both directions, such as the Walsh–Hadamard transform.

In the applications that now follow, f will always be the indicator function of a family $\mathcal{F}$. In this case having $\operatorname{supp}(f)$ quickly available translates to $\mathcal{F}$ being efficiently listable; for example, with polynomial delay.

2.3. Covers

The easiest application of the trimmed Moebius inversion computes for each $X \in \uparrow\mathcal{F}$ the cover number $c(X)$. This is a particularly straightforward function of the zeta transform of the indicator function f: simply raise each element of $f\zeta$ to the kth power and transform the result back using μ. To see this, observe that both sides of the equation

$$(c\zeta)(Y) = \big((f\zeta)(Y)\big)^k \tag{2.9}$$

count the number of ways to choose k-tuples $(S_1, \dots, S_k)$ with $S_i \subseteq Y$ and $S_i \in \mathcal{F}$. By Moebius inversion, we can recover c by applying μ to both sides of (2.9).

Algorithm C. (*Cover number.*) Computes $c(X)$ for all $X \in \uparrow\mathcal{F}$ given $\mathcal{F}$. The sets $\mathcal{L}(r)$ and auxiliary values $g_j(X)$ are as in Algorithm Z; also required are auxiliary values $h_j(X)$ for Moebius transform.

C1: For each $X \in \mathcal{F}$, insert X into $\mathcal{L}(|X|)$. Set the current rank $r = 0$.

C2: Select any $X \in \mathcal{L}(r)$ and remove it from $\mathcal{L}(r)$.

C3: [Zeta transform.] Set $g_0(X) = [X \in \mathcal{F}]$. For each $j = 1, \dots, n$, set

$$g_j(X) = [j \in X]g_{j-1}(X \setminus \{j\}) + g_{j-1}(X)\,.$$

[At this point it holds that $g_n(X) = (f\zeta)(X)$.]

C4: [Evaluate zeta transform of $c(X)$.] Set $h_0(X) = g_n(X)^k$.

C5: [Moebius transform.] For each $j = 1, \dots, n$, set

$$h_j(X) = -[j \in X]h_{j-1}(X \setminus \{j\}) + h_{j-1}(X)\,.$$

C6: Output X and $h_n(X)$.

C7: For each $j \notin X$, insert $X \cup \{j\}$ into $\mathcal{L}(r+1)$.

C8: If $\mathcal{L}(r)$ is empty, then increment $r \leq n$ until $\mathcal{L}(r)$ is nonempty; terminate if $r = n$ and $\mathcal{L}(n)$ is empty.

C9: Go to C2.

2.4. Partitions

What makes the partition problem slightly less transparent is the fact that we need to use dynamic programming to assemble partitions from sets with different ranks. To this end, we need to compute for each rank s the 'ranked zeta transform'

$$(f\zeta^{(s)})(X) \;=\; \sum_{Y \subseteq X, |Y|=s} f(Y)\,.$$

For rank s, consider the number $d^{(s)}(Y)$ of tuples $(S_1, \dots, S_k)$ with $S_i \in \mathcal{F}$, $S_i \subseteq Y$, $S_1 \cup \dots \cup S_k = Y$ and $|S_1| + \dots + |S_k| = s$. Then $d(Y) = d^{(|Y|)}(Y)$. Furthermore, the

zeta-transform $(d^{(s)}\zeta)(X)$ counts the number of ways to choose $(S_1, \ldots, S_k)$ with $S_i \subseteq X$, $S_i \in \mathcal{F}$, and $|S_1| + \cdots + |S_k| = s$. Another way to count the exact same quantity is

$$q(k, s, X) \;=\; \sum_{s_1+\cdots+s_k=s} \prod_{i=1}^{k} (f\zeta^{(s_i)})(X)\,. \tag{2.10}$$

Thus we can recover $d^{(s)}(Y)$ from $q(k, s, X)$ by Moebius inversion.

As it stands, (2.10) is time-consuming to evaluate even given all the ranked zeta transforms, but we can compute it efficiently using dynamic programming based on the recurrence

$$q(k, s, X) = \begin{cases} \sum_{t=0}^{s} q(k-1, s-t, X)(f\zeta^{(t)})(X), & \text{if } k > 1\,, \\ (f\zeta^{(s)})(X), & \text{if } k = 1\,. \end{cases}$$

This happens in Step D4.

Algorithm D. (*Disjoint cover number.*) Computes $d(X)$ for all $X \in {\uparrow}\mathcal{F}$ given $\mathcal{F}$. The sets $\mathcal{L}(r)$ are as in Algorithm Z; we also need auxiliary values $g_j^{(s)}(X)$ and $h_j^{(s)}(X)$ for all $X \in {\uparrow}\mathcal{F}$, $1 \le j \le n$, and $0 \le s \le n$; it holds that $g_n^{(s)}(X) = (f\zeta^{(s)})(X)$ and $h_n^{(s)}(X) = d^{(s)}(X)$.

D1: For each $X \in \mathcal{F}$, insert X into $\mathcal{L}(|X|)$. Set the current rank $r = 0$.

D2: Select any $X \in \mathcal{L}(r)$ and remove it from $\mathcal{L}(r)$.

D3: [Ranked zeta transform.] For each $s = 0, \ldots, n$, set $g_0^{(s)}(X) = [X \in \mathcal{F}][|X| = s]$. For each $j = 1, \ldots, n$ and $s = 0, \ldots, n$, set

$$g_j^{(s)}(X) = [j \in X] g_{j-1}^{(s)}(X \setminus \{j\}) + g_{j-1}^{(s)}(X)\,.$$

[At this point it holds that $g_n^{(s)}(X) = (f\zeta^{(s)})(X)$ for all $0 \le s \le n$.]

D4: [Evaluate zeta transform of $d^{(s)}$.] For each $s = 0, \ldots, n$, set $q(1, s) = g_n^{(s)}(X)$. For each $i = 2, \ldots, k$ and $s = 0, \ldots, n$, set $q(i, s) = \sum_{t=0}^{s} q(i-1, s-t) g_n^{(t)}(X)$.

D5: [Ranked Moebius transform.] For each $s = 0, \ldots, n$, set $h_0^{(s)}(X) = q(k, s)$. For each $j = 1, \ldots, n$ and $s = 0, \ldots, n$, set

$$h_j^{(s)}(X) = -[j \in X] h_{j-1}^{(s)}(X \setminus \{j\}) + h_{j-1}^{(s)}(X)\,.$$

[At this point it holds that $h_n^{(s)}(X) = d^{(s)}(X)$ for all $0 \le s \le n$.]

D6: Output X and $h_n^{(|X|)}(X)$.

D7: For each $j \notin X$, insert $X \cup \{j\}$ into $\mathcal{L}(r+1)$.

D8: If $\mathcal{L}(r)$ is empty, then increment $r \le n$ until $\mathcal{L}(r)$ is nonempty; terminate if $r = n$ and $\mathcal{L}(n)$ is empty.

D9: Go to D2.

2.5. Packings

According to (2.3), to compute $p(X)$ it suffices to zeta-transform the partition number. This amounts to running Algorithm Z after Algorithm D. (For a different approach, see [4].)

3. Applications

3.1. The number of dominating sets in sparse graphs

This section is purely combinatorial. Let $\mathcal{D}$ denote the dominating sets of a graph. A complete graph has $2^n - 1$ dominating sets, and sparse graphs can have almost as many: the n-star graph has 2^{n-1} dominating sets and average degree less than 2. Thus we ask how large $|\mathcal{D}|$ can be for graphs with bounded maximum degree. An easy example is provided by the disjoint union of complete graphs of order $\Delta + 1$: every vertex subset that includes at least one vertex from each component is dominating, so $|\mathcal{D}| = (2^{\Delta+1} - 1)^{n/(\Delta+1)}$. We shall show that this is in fact the largest possible $\mathcal{D}$ for graphs of maximum degree Δ. Our analysis is based on the following intersection theorem.

Lemma 3.1 (Chung *et al.* [8])**.** *Let U be a finite set with subsets $P_1, \ldots, P_m$ such that every $u \in U$ is contained in at least δ subsets. Let $\mathcal{F}$ be a family of subsets of U. For each $1 \le \ell \le m$, define the projections $\mathcal{F}_\ell = \{ F \cap P_\ell : F \in \mathcal{F} \}$. Then*

$$|\mathcal{F}|^\delta \le \prod_{\ell=1}^{m} |\mathcal{F}_\ell| \,.$$

Theorem 3.2. *The number of dominating sets of an n-vertex graph with maximum degree Δ is at most $(2^{\Delta+1} - 1)^{n/(\Delta+1)}$.*

Proof. Let $G = (V, E)$ be a graph with $|V| = n$ and maximum degree Δ. For each $v \in V$, let A_v be the closed neighbourhood around vertex v,

$$A_v = \{v\} \cup \{ u \in V : uv \in E \} \,. \tag{3.1}$$

Next, for each $u \in V$ with degree $d(u) < \Delta$, add u to $\Delta - d(u)$ of the sets A_v not already containing u (it does not matter which). Let $a_v = |A_v|$ and note that $\sum_v a_v = (\Delta + 1)n$.

We want to apply Lemma 3.1. To this end, let $U = V$ and $m = n$. By construction, every $u \in V$ belongs to exactly $\delta = \Delta + 1$ subsets A_v. To get a nontrivial bound on $\mathcal{D}$ we need to bound the size of $\mathcal{D}_v = \{ D \cap A_v : D \in \mathcal{D} \}$. Every $D \cap A_v$ is one of the 2^{a_v} subsets of A_v, but none of the $D \cap A_v$ can be the empty set, because either v or one of its neighbours must belong to the dominating set D. Thus $|\mathcal{D}_v| \le 2^{a_v} - 1$. By Lemma 3.1, we have

$$|\mathcal{D}|^{\Delta+1} \le \prod_v (2^{a_v} - 1) \,. \tag{3.2}$$

Since $x \mapsto \log(2^x - 1)$ is concave, Jensen's inequality gives

$$\frac{1}{n} \sum_v \log(2^{a_v} - 1) \le \log(2^{\sum_v a_v/n} - 1) = \log(2^{\Delta+1} - 1) \,.$$

Taking exponentials and combining with (3.2) gives $|\mathcal{D}|^{\Delta+1} \le (2^{\Delta+1} - 1)^n$. ∎

3.2. Domatic Number

We first observe that a graph can be packed with k dominating sets if and only if it can be packed with k *minimal* dominating sets, so we can consider k-packings from $\min \mathcal{D}$ instead of $\mathcal{D}$. This has the advantage that $\min \mathcal{D}$ can be listed faster than 2^n.

Lemma 3.3 (Fomin *et al.* [10])**.** *Any n-vertex graph has at most $O^*(1.7170^n)$ minimal dominating sets, and they can be listed within that time bound.*

Theorem 3.4. *For an n-vertex graph G with maximum degree Δ we can decide in time*

$$O^*\big((2^{\Delta+1}-2)^{n/(\Delta+1)}\big)$$

whether G admits a packing with k dominating sets.

Proof. We use Algorithm D with $\mathcal{F} = \min \mathcal{D}$. By the above lemma, we can complete Step D1 in time $O^*(1.7170^n)$. The rest of the algorithm requires time $O^*(|{\uparrow}\min \mathcal{D}|)$. Since every superset of a dominating set is itself dominating, ${\uparrow}\min \mathcal{D}$ is a sub-family of $\mathcal{D}$ (in fact, it is exactly $\mathcal{D}$), so Theorem 3.2 bounds the total running time by

$$O^*\big((2^{\Delta+1}-1)^{n/(\Delta+1)}\big)\,.$$

We can do slightly better if we modify Algorithm D in Step D7 to insert $X \cup \{j\}$ only if it excludes at least one vertex for each closed neighbourhood. Put otherwise, we insert $X \cup \{j\}$ only if the set $V \setminus (X \cup \{j\})$ dominates the graph G. The graph then has Domatic Number at least $k+1$ if and only if the algorithm reports some X for which $d(X)$ is nonzero. The running time can again be bounded as in Theorem 3.2 but now $D \cap A_v$ can neither be the empty set, nor be equal to A_v. Thus the application of Lemma 3.1 can be strengthened to yield the claimed result. ∎

3.3. Chromatic Number

Our first argument for Chromatic Number is similar; we give a stronger and slightly more complicated argument in §3.4.

We consider the independent sets $\mathcal{I}$ of a graph. An independent set is not necessarily dominating, but it is easy to see that a *maximal* independent set is dominating. Moreover, the Moon–Moser bound tells us they are few, and Tsukiyama *et al.* tell us how to list them with polynomial delay:

Lemma 3.5 (Moon and Moser [15]; Tsukiyama *et al.* [16])**.** *Any n-vertex graph has at most $O^*(1.4423^n)$ maximal independent sets, and they can be listed within that bound.*

Theorem 3.6. *For an n-vertex graph G with maximum degree Δ we can decide in time*

$$O^*\big((2^{\Delta+1}-1)^{n/(\Delta+1)}\big)$$

whether G admits a covering with k independent sets.

Proof. It is easy to see that G can be covered with k independent sets if and only if it can be covered with k *maximal* independent sets, so we will use Algorithm C on $\max \mathcal{I}$. Step C1 is completed in time $O^*(1.4423^n)$, and the rest of the algorithm considers only the points in ${\uparrow}\max \mathcal{I}$, which all belong to $\mathcal{D}$. Again, Theorem 3.2 bounds the total running time. ∎

3.4. Chromatic Number via bipartite subgraphs

We can do somewhat better by considering the family $\mathcal{B}$ of vertex sets of induced *bipartite* subgraphs, that is, the family of sets $B \subseteq V$ for which the induced subgraph $G[B]$ is bipartite. As before, the literature provides us with a nontrivial listing algorithm:

Lemma 3.7 (Byskov and Eppstein [7])**.** *Any n-vertex graph has at most $O^*(1.7724^n)$ maximal induced bipartite subgraphs, and they can be listed within that bound.*

The family $\max \mathcal{B}$ is more than just dominating, which allows us to use Lemma 3.1 in a stronger way.

Theorem 3.8. *For an n-vertex graph of maximum degree Δ it holds that*

$$|{\uparrow}\max \mathcal{B}| \leq (2^{\Delta+1} - \Delta - 1)^{n/(\Delta+1)}\,.$$

Proof. Let $G = (V, E)$ be a graph with $|V| = n$ and maximum degree Δ. Let $\mathcal{F} = {\uparrow}\max \mathcal{B}$. Let A_v be as in (3.1). With the objective of applying Lemma 3.1, we need to bound the number of sets in $\mathcal{F}_v = \{\, F \cap A_v : F \in \mathcal{F} \,\}$.

Assume first that G is Δ-regular. Let $A_v = \{v, u_1, \ldots, u_\Delta\}$. We will rule out $\Delta + 1$ candidates for $F \cap A_v$, namely

$$\emptyset, \{u_1\}, \ldots, \{u_\Delta\} \notin \mathcal{F}_v\,. \tag{3.3}$$

This then shows that $|\mathcal{F}_v| \leq 2^{\Delta+1} - \Delta - 1$ and thus the bound follows from Lemma 3.1.

To see that (3.3) holds, observe that $F \in \mathcal{F}$ contains a $B \subseteq F$ such that the induced subgraph $G[B]$ is maximal bipartite. To reach a contradiction, assume that there exists a $v \in V$ with $F \cap A_v \subseteq \{u_\ell\}$. Since $B \subseteq F$, we have $B \cap A_v \subseteq \{u_\ell\}$, implying that v does not belong to B, and that at most one of its neighbours does. Consequently, $G[B \cup \{v\}]$ is also bipartite, and v belongs to a partite set opposite to any of its neighbours. This contradicts the fact that $G[B]$ is maximal bipartite.

To establish the non-regular case, we can proceed as in the proof of Theorem 3.2, adding each $u \in V$ with $d(u) < \Delta$ to some $\Delta - d(u)$ of the sets A_v not already including u. Note that by adding y new vertices to A_v originally containing x vertices, we get $|\{F \cap A_v : F \in \mathcal{F}\}| \leq 2^y(2^x - x - 1)$. Next, since $2^y(2^x - x - 1) \leq 2^{y+x} - (y + x) - 1$ for all non-negative integers y, x and $\log(2^x - x - 1)$ is a concave function, the bound follows as before via Jensen's inequality. ∎

Theorem 3.9. *For an n-vertex graph G with maximum degree Δ we can decide in time*

$$O\big((2^{\Delta+1} - \Delta - 1)^{n/(\Delta+1)}\big)$$

whether G admits a covering with k independent sets.

Proof. When k is even, it is easy to see that G can be covered by k independent sets if and only if it can be covered by $k' = k/2$ maximal bipartite sets, so we will use Algorithm C on $\max \mathcal{B}$ and investigate whether $c(V) \neq 0$.

When k is odd, we again use Algorithm C with $k' = (k-1)/2$ maximal bipartite sets, but this time we check whether an X is output such that both $c(X) \neq 0$ and $V \setminus X$ is independent in G.

In both cases the running time bound follows from Theorem 3.8. ∎

4. Concluding Remarks

Since the presented improvements on running time bounds are modest, one can ask whether this is because of weak bounds or because of inherent limitations of the technique. We observe that the running time bounds in Theorems 3.4, 3.6, and 3.9 are met by a disjoint union of complete graphs of order $\Delta + 1$. Thus, either further trimming or splitting into connected components is required for improved algorithms in this context.

We chose to demonstrate the technique for Chromatic and Domatic Number since these are well-known and well-studied. To briefly demonstrate some further application potential, more artificial problem variants such as determining if a Δ-regular graph has domatic number at least $\Delta/2$, or if the square of a Δ-regular graph has chromatic number at most

$3\Delta/2$, admit stronger bounds. For example, if G has domatic number at least d, d even, then its vertices can be partitioned into two sets, both of which contain $d/2$ dominating sets. This suggests the following meet-in-the-middle strategy. Run Algorithm D with $\mathcal{F}$ equal to all dominating sets and $k = d/2$, but modify Step D7 to insert $X \cup \{j\}$ only if $|A_v \setminus (X \cup \{j\})| \geq d/2$ holds for all vertices v. At termination, we check whether the algorithm has output two sets X and Y such that $X \cup Y = V$ and $d(X), d(Y) > 0$. (For example, one can check for duplicates in a table with entry $\{X, V \setminus X\}$ for each output X with $d(X) > 0$.) This algorithm variant considers only sets with many forbidden intersections with the neighbourhoods of vertices, which translates into stronger bounds via Lemma 3.1.

References

[1] R. Beigel and D. Eppstein, 3-*coloring in time* $O(1.3289^n)$, J. Algorithms **54** (2005), 168–204.

[2] A. Björklund and T. Husfeldt, *Exact algorithms for exact satisfiability and number of perfect matchings*, Algorithmica, to appear. Prelim. version in Proc. 33rd International Colloquium on Automata, Languages and Programming, Lect. Notes in Comp. Science, Vol. 4051, Springer, Berlin, 2006, pp. 548–559.

[3] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto, *Fourier meets Möbius: fast subset convolution*, Proc. 39th ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, NY, 2007, pp. 67–74.

[4] A. Björklund, T. Husfeldt, and M. Koivisto, *Set partitioning via inclusion–exclusion*, SIAM J. Comput., to appear. Prelim. versions in Proc. 47th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 2006, pp. 575–582, 583–590.

[5] R. L. Brooks, *On colouring the nodes of a network*, Proc. Cambridge Philos. Soc. **37** (1941), 194–197.

[6] J. M. Byskov, *Enumerating maximal independent sets with applications to graph colouring*, Oper. Res. Lett. **32** (2004), 547–556.

[7] ———, *Exact algorithms for graph colouring and exact satisfiability*, Ph.D. Thesis, Univ. Aarhus, 2004.

[8] F. R. K. Chung, P. Frankl, R. L. Graham, and J. B. Shearer, *Some intersection theorems for ordered sets and graphs*, J. Combin. Theory Ser. A **43** (1986), 23–37.

[9] D. Eppstein, *Small maximal independent sets and faster exact graph coloring*, J. Graph Algorithms Appl. **7** (2003), 131–140.

[10] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov, *Bounding the number of minimal dominating sets: a measure and conquer approach*, Proc. 16th Intern. Symposium on Algorithms and Computation, Lect. Notes in Comp. Science, Vol. 3827, Springer, Berlin, 2005, pp. 573–582.

[11] F. V. Fomin, S. Gaspers, S. Saurabh, and A. A. Stepanov, *On two techniques of combining branching and treewidth*, Algorithmica, to appear. Reports in Informatics, no. 337, Department of Informatics, University of Bergen, 2006.

[12] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith, *Algorithms based on the treewidth of sparse graphs*, Revised Selected Papers from the 31st Intern. Workshop on Graph-Theoretic Concepts in Computer Science, Lect. Notes in Comp. Science, Vol. 3787, Springer, Berlin, 2005, pp. 385–396.

[13] D. E. Knuth, *The Art of Computer Programming, Vol.* 2: *Seminumerical Algorithms*, 3rd ed., Addison–Wesley, Reading, MA, 1997.

[14] E. L. Lawler, *A note on the complexity of the chromatic number problem*, Inform. Process. Lett. **5** (1976), 66–67.

[15] J. W. Moon and L. Moser, *On cliques in graphs*, Israel J. Math. **3** (1965), 23–28.

[16] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, *A new algorithm for generating all the maximal independent sets*, SIAM J. Comput. **6** (1977), 505–517.

[17] F. Yates, *The design and analysis of factorial experiments*, Technical Communication 35, Commonwealth Bureau of Soils, Harpenden, U.K., 1937.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 97-108
www.stacs-conf.org

ON THE COMPLEXITY OF THE INTERLACE POLYNOMIAL

MARKUS BLÄSER [1] AND CHRISTIAN HOFFMANN [1]

[1] Saarland University, Computer Science, Postfach 151150, 66041 Saarbrücken, Germany
E-mail address, Markus Bläser: mblaeser@cs.uni-sb.de
E-mail address, Christian Hoffmann: hoffmann@cs.uni-sb.de

Abstract. We consider the two-variable interlace polynomial introduced by Arratia, Bollobás and Sorkin (2004). We develop two graph transformations which allow us to derive point-to-point reductions for the interlace polynomial. Exploiting these reductions we obtain new results concerning the computational complexity of evaluating the interlace polynomial at a fixed point. Regarding *exact* evaluation, we prove that the interlace polynomial is #P-hard to evaluate at every point of the plane, except at one line, where it is trivially polynomial time computable, and four lines and two points, where the complexity mostly is still open. This solves a problem posed by Arratia, Bollobás and Sorkin (2004). In particular, we observe that three specializations of the two-variable interlace polynomial, the vertex-nullity interlace polynomial, the vertex-rank interlace polynomial and the independent set polynomial, are almost everywhere #P-hard to evaluate, too. For the independent set polynomial, our reductions allow us to prove that it is even hard to *approximate* at every point except at -1 and 0.

1. Introduction

The number of Euler circuits in specific graphs and their interlacings turned out to be a central issue in the solution of a problem related to DNA sequencing by hybridization [ABCS00]. This led to the definition of a new graph polynomial, the one-variable interlace polynomial [ABS04a]. Further research on this polynomial inspired the definition of a two-variable interlace polynomial $q(G; x, y)$ containing as special cases the following graph polynomials: $q_N(G; y) = q(G; 2, y)$ is the original one-variable interlace polynomial which was renamed to "vertex-nullity interlace polynomial", $q_R(G; x) = q(G; x, 2)$ is the new "vertex-rank interlace polynomial" and $I(G; x) = q(G; 1, 1 + x)$ is the independent set polynomial[1] [ABS04b].

Although the interlace polynomial $q(G; x, y)$ is a different object from the celebrated Tutte polynomial (also known as dichromatic polynomial, see, for instance, [Tut84]), they are also similar to each other. While the Tutte polynomial can be defined recursively

Key words and phrases: computational complexity, approximation, interlace polynomial, independent set polynomial, graph transformation.

[1]The independent set polynomial of a graph G is defined as $I(G; x) = \sum_{j \geq 0} i(G; j)x^j$, where $i(G; j)$ denotes the number of independent sets of cardinality j of G.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

by a deletion-contraction identity on edges, the interlace polynomial satisfies recurrence relations involving several operations on vertices (deletion, pivotization, complementation).

Besides the deletion-contraction identity, the so called state expansion is a well-known way to define the Tutte polynomial. Here the similarity to the two-variable interlace polynomial is especially striking: while the interlace polynomial is defined as a sum over all vertex subsets of the graph using the rank of adjacency matrices (see (2.1)), the state expansion of the Tutte polynomial can be interpreted as a sum over all edge subsets of the graph using the rank of incidence matrices (see (4.1)) [ABS04b, Section 1].

References to further work on the interlace polynomial can be found in [ABS04b] and [EMS06].

1.1. Previous work

The aim of this paper is to explore the computational complexity of evaluating[2] the two-variable interlace polynomial $q(G; x, y)$. For the Tutte polynomial this problem was solved in [JVW90]: Evaluating the Tutte polynomial is #P-hard at any algebraical point of the plane, except on the hyperbola $(x-1)(y-1) = 1$ and at a few special points, where the Tutte polynomial can be evaluated in polynomial time. For the two-variable interlace polynomial $q(G; x, y)$, only on a one-dimensional subset of the plane (on the lines $x = 2$ and $x = 1$) some results about the evaluation complexity are known.

A connection between the vertex-nullity interlace polynomial and the Tutte polynomial of planar graphs [ABS04a, End of Section 7], [EMS06, Theorem 3.1] shows that evaluating q is #P-hard almost everywhere on the line $x = 2$ (Corollary 4.4).

It has also been noticed that $q(G; 1, 2)$ evaluates to the number of independent sets of G [ABS04b, Section 5], which is #P-hard to compute [Val79]. Recent work on the matching generating polynomial [AM07] implies that evaluating q is #P-hard almost everywhere on the line $x = 1$ (Corollary 4.10).

A key ingredient of [JVW90] is to apply graph transformations known as stretching and thickening of edges. For the Tutte polynomial, these graph transformations allow us to reduce the evaluation at one point to the evaluation at another point. For the interlace polynomial no such graph transformations have been given so far.

1.2. Our results

We develop two graph transformations which are useful for the interlace polynomial: cloning and combing of vertices. Applying cloning or combing allows us to reduce the evaluation of the interlace polynomial at some point to the evaluation of it at another point, see Theorem 3.3 and Theorem 3.5. We exploit this to obtain the following new results about the computational complexity of $q(G; x, y)$.

We prove that the two-variable interlace polynomial $q(G; x, y)$ is #P-hard to evaluate at almost every point of the plane, Theorem 4.12, see also Figure 1. Even though there are some unknown (gray, in Figure 1) lines left on the complexity map for q, this solves a challenge posed in [ABS04b, Section 5]. In particular we obtain the new result that

[2]See Section 2.2 for a precise definition.

evaluating the vertex-rank interlace polynomial $q_R(G;x)$ is #P-hard at almost every point (Corollary 4.13). Our techniques also give a new proof that the independent set polynomial is #P-hard to evaluate almost everywhere (Remark 4.11).

Apart from these results on the computational complexity of evaluating the interlace polynomial *exactly*, we also show that the values of the independent set polynomial (which is the interlace polynomial $q(G;x,y)$ on the line $x=1$) are hard to *approximate* almost everywhere (Theorem 5.4).

2. Preliminaries

2.1. Interlace Polynomials

We consider undirected graphs without multiple edges but with self loops allowed. Let $G=(V,E)$ be such a graph and $A \subseteq V$. By $G[A]$ we denote $(A, \{e | e \in E, e \subseteq A\})$, the subgraph of G induced by A. The adjacency matrix of G is the symmetric $n \times n$-matrix $M=(m_{ij})$ over $\mathbb{F}_2=\{0,1\}$ with $m_{i,j}=1$ iff $\{i,j\} \in E$. The rank of this matrix is its rank over $\mathbb{F}_2$. Slightly abusing notation we write $rk(G)$ for this rank. This allows us to define the two-variable interlace polynomial.

Definition 2.1 ([ABS04b])**.** Let $G=(V,E)$ be an undirected graph. The interlace polynomial $q(G;x,y)$ of G is defined as

$$q(G;x,y) = \sum_{A \subseteq V} (x-1)^{rk(G[A])}(y-1)^{|A|-rk(G[A])}. \tag{2.1}$$

In Section 3 we will introduce graph transformations (graph cloning and graph combing) which perform one and the same operation (cloning one single vertex, adding a comb to one single vertex, resp.) on every vertex of a graph. Instead of relating the interlace polynomial of the original graph directly to the interlace polynomial of the transformed graph, we will analyze how, say, cloning *one single vertex* changes the interlace polynomial. To express this, we must be able to treat the vertex being cloned in a particular way, differently from the other vertices. This becomes possible using a *multivariate* version of the interlace polynomial, in which each vertex has its own variable. Once we can express the effect of cloning *one* vertex by an appropriate substitution of the vertex variable in the multivariate interlace polynomial, cloning *all* the vertices amounts to a simple substitution of all vertex variables and brings us back to a bivariate interlace polynomial. This procedure has been applied successfully to the Tutte polynomial [Sok05, BM06].

We choose the following multivariate interlace polynomial, which is similar to the multivariate Tutte polynomial of Sokal [Sok05] and a specialization of the multivariate interlace polynomial defined by Courcelle [Cou07].

Definition 2.2. Let $G=(V,E)$ be an undirected graph. For each $v \in V$ let x_v be an indeterminate. Writing x_A for $\prod_{v \in A} x_v$, we define the following multivariate interlace polynomial:

$$P(G;u,\mathbf{x}) = \sum_{A \subseteq V} x_A u^{rk(G[A])}.$$

Substituting each x_v in $P(G;u,\mathbf{x})$ by x, we obtain another bivariate interlace polynomial:

$$P(G;u,x) = \sum_{A\subseteq V} x^{|A|} u^{rk(G[A])}.$$

An easy calculation proves that q and P are closely related:

Lemma 2.3. *Let G be a graph. Then we have the polynomial identities $q(G;x,y) = P(G;\frac{x-1}{y-1}, y-1)$ and $P(G;u,x) = q(G;ux+1,x+1)$.* ∎

2.2. Evaluating Graph Polynomials

Given $\xi, \upsilon \in \mathbb{Q}$ we want to analyze the following computational problem:

Input: Graph G
Output: $q(G;\xi,\upsilon)$

This is what we mean by "evaluating the interlace polynomial q at the point (ξ,υ)". As an abbreviation for this computational problem we write

$$q(\xi,\upsilon),$$

which should not be confused with the expression $q(G;\xi,\upsilon)$ denoting just a value in $\mathbb{Q}$. Evaluating other graph polynomials such as P, q_N, q_R and I is defined accordingly.

If P_1 and P_2 are computational problems we use $P_1 \preceq_T P_2$ ($P_1 \preceq^{\mathtt{m}} P_2$) to denote a polynomial time Turing reduction (polynomial time many-one reduction, resp.) from P_1 to P_2. For instance, Lemma 2.3 gives

Corollary 2.4. *For $\xi,\upsilon \in \tilde{\mathbb{Q}}$, $\upsilon \neq 1$, we have $q(\xi,\upsilon) \preceq^{\mathtt{m}} P(\frac{\xi-1}{\upsilon-1}, \upsilon-1)$. For $\mu,\xi \in \tilde{\mathbb{Q}}$ we have $P(\mu,\xi) \preceq^{\mathtt{m}} q(\mu\xi+1,\xi+1)$.* ∎

Here $\tilde{\mathbb{Q}}$ denotes some finite dimensional field extension $\mathbb{Q} \subseteq \tilde{\mathbb{Q}} \subseteq \mathbb{R}$, which has a discrete representation. As $\sqrt{2}$ will play an important role but we are not able to use arbitrary real numbers as the input for a Turing machine, we use $\tilde{\mathbb{Q}}$ instead of $\mathbb{Q}$ or $\mathbb{R}$. We fix some $\tilde{\mathbb{Q}}$ for the rest of this paper. This construction is done in the spirit of Jaeger, Vertigan, and Welsh [JVW90] who also propose to adjoin a finite number of points to $\mathbb{Q}$ in order to talk about the complexity at irrational points. To some extent, this is an ad hoc construction, but it is sufficient for this work.

3. Graph Transformations for the Interlace Polynomial

Now we describe our graph transformations, the cloning and combing of vertices. The main results of this section are Theorem 3.3 and Theorem 3.5 which describe the effect of cloning and combing on the interlace polynomial.

3.1. Cloning

Cloning vertices in the graph yields our first graph transformation.

Cloning one vertex. Let $G = (V, E)$ be a graph. Let $a \in V$ be some vertex (the one which will be cloned) and N the set of neighbors of a, $V' = V \setminus \{a\}$ and $M = V' \setminus N$. The graph G with a cloned, G_{aa}, is obtained out of G in the following way: Insert a new isolated vertex a'. Connect a' to all vertices in N. If a does not have a self loop, we are done. Otherwise connect a and a' and insert a self loop at a'. Thus, adjacency matrices of the original (cloned, resp.) graph are

$$B = \begin{array}{c|ccc} & a & N & M \\ \hline a & b & \mathbf{1} & \mathbf{0} \\ N & \mathbf{1} & A_{11} & A_{12} \\ M & \mathbf{0} & A_{21} & A_{22} \end{array} \quad \text{and} \quad B_{aa} = \begin{array}{c|cccc} & a' & a & N & M \\ \hline a' & b & b & \mathbf{1} & \mathbf{0} \\ a & b & b & \mathbf{1} & \mathbf{0} \\ N & \mathbf{1} & \mathbf{1} & A_{11} & A_{12} \\ M & \mathbf{0} & \mathbf{0} & A_{21} & A_{22} \end{array}, \text{ resp,} \tag{3.1}$$

where $b = 1$ if a has a self loop and $b = 0$ otherwise. As the first column of B_{aa} equals its second column, as well as the first row equals the second row, we can remove the first row and the first column of B_{aa} without changing the rank. This also holds when we consider the adjacency matrices of $G[A]$ ($G_{aa}[A]$, resp.) instead of G (G_{aa} resp.) for $A \subseteq V'$. Thus we have for any $A \subseteq V'$

$$rk(G_{aa}[A]) = rk(G[A]), \tag{3.2}$$

$$rk(G_{aa}[A \cup \{a, a'\}]) = rk(G_{aa}[A \cup \{a\}]) = rk(G_{aa}[A \cup \{a'\}]) = rk(G[A \cup \{a\}]). \tag{3.3}$$

Let $\mathbf{x} = (x_v)_{v \in V(G_{aa})}$ be a labeling of the vertices of G_{aa} by indeterminates. Define $\mathbf{X}$ to denote the following labeling of the vertices of G: $X_v := x_v$ for all $v \in V'$, $X_a := (1 + x_a)(1 + x_{a'}) - 1 = x_a + x_{a'} + x_a x_{a'}$. Then we have

Lemma 3.1. *$P(G_{aa}; u, \mathbf{x}) = P(G; u, \mathbf{X})$.*

Proof. On the one hand we have

$$\begin{aligned} &P(G_{aa}; u, \mathbf{x}) \\ &= \sum_{A \subseteq V'} x_A (u^{rk(G_{aa}[A])} + x_a u^{rk(G_{aa}[A \cup \{a\}])} + x_{a'} u^{rk(G_{aa}[A \cup \{a'\}])} + x_a x_{a'} u^{rk(G_{aa}[A \cup \{a, a'\}])}) \\ &= \sum_{A \subseteq V'} x_A (u^{rk(G[A])} + (x_a + x_{a'} + x_a x_{a'}) u^{rk(G[A \cup \{a\}])}) \text{ by (3.2), (3.3).} \end{aligned}$$

On the other hand we have

$$\begin{aligned} P(G; u, \mathbf{X}) &= \sum_{A \subseteq V'} X_A (u^{rk(G[A])} + X_a u^{rk(G[A \cup \{a\}])}) \\ &= \sum_{A \subseteq V'} x_A (u^{rk(G[A])} + (x_a + x_{a'} + x_a x_{a'}) u^{rk(G[A \cup \{a\}])}). \end{aligned}$$

■

Cloning all vertices. Fix some k. Given a graph G, the graph G_k is obtained by cloning each vertex of G exactly $k-1$ times. Note that the result of the cloning is independent of the order in which the different vertices are cloned. For $a \in V(G)$ let $a_1, \ldots, a_k$ be the corresponding vertices in G_k. For a vertex labeling $\mathbf{x}$ of G_k we define the vertex labeling $\mathbf{X}$ of G by $X_a = (1+x_{a_1})(1+x_{a_2})\cdots(1+x_{a_k})-1$ for $a \in V(G)$. Applying Lemma 3.1 repeatedly we obtain

Lemma 3.2. $P(G_k; u, \mathbf{x}) = P(G; u, \mathbf{X})$. ∎

Substitution of x_v by x for all vertices v gives

Theorem 3.3. *Let G be a graph and G_k be obtained out of G by cloning each vertex of G exactly $k-1$ times. Then*

$$P(G_k; u, x) = P(G; u, (1+x)^k - 1). \tag{3.4}$$

∎

As we will use it in the proof of Theorem 4.12, we note the following identity for q, which can be derived from Theorem 3.3 using Lemma 2.3:

$$q(G_k; x, y) = q(G; (x-1)\frac{y^k-1}{y-1} + 1, y^k). \tag{3.5}$$

Theorem 3.3 also implies the following reduction for the interlace polynomial, which is the foundation for our results in Section 4.

Proposition 3.4. *Let $B_2 = \{0, -1, -2\}$ and x be an indeterminate. For $\mu \in \tilde{\mathbb{Q}}, \xi \in \tilde{\mathbb{Q}} \setminus B_2$ we have $P(\mu, x) \preceq_T P(\mu, \xi)$. (For any $\mu \in \tilde{\mathbb{Q}}$, we write $P(\mu, x)$ to denote the following computational problem: given a graph G compute $P(G; \mu, x)$, which is a polynomial in x with coefficients in $\tilde{\mathbb{Q}}$.)*

Proof. Let μ and ξ be given such that they fulfill the precondition of the proposition. Given a graph $G =: G_1$ with n vertices, we build $G_2, G_3, \ldots, G_{n+1}$, where G_i is obtained out of G by cloning each vertex $i-1$ times. This is possible in time polynomial in n. By Theorem 3.3, a call to an oracle for $P(\mu, \xi)$ with input G_i gives us $P(G; \mu, (1+\xi)^i - 1)$ for $i = 1, \ldots, n+1$. The restriction on ξ guarantees that for $i = 1, 2, 3, \ldots$ the expression $(1+\xi)^i - 1$ evaluates to pairwise different values. Thus, for $P(G; \mu, x)$, which is a polynomial in x of degree $\leq n$, we have obtained the values at $n+1$ distinct points. Using Lagrange interpolation we determine the coefficients of $P(G; \mu, x)$. ∎

3.2. Combs

The comb transformation sometimes helps, when cloning has not the desired effect. Let $G = (V, E)$ be a graph and $a \in V$ some vertex. Then we define the k-comb of G at a as $G_{a,k} = (V \cup \{a_1, \ldots, a_k\}, E \cup \{\{a, a_1\}, \ldots, \{a, a_k\}\})$, with $a_1, \ldots, a_k$ being new vertices.

Using similar arguments as with vertex cloning, combing of vertices yields a point-to-point reduction for the interlace polynomial, too. The proof of the following theorem can be found in [BH07].

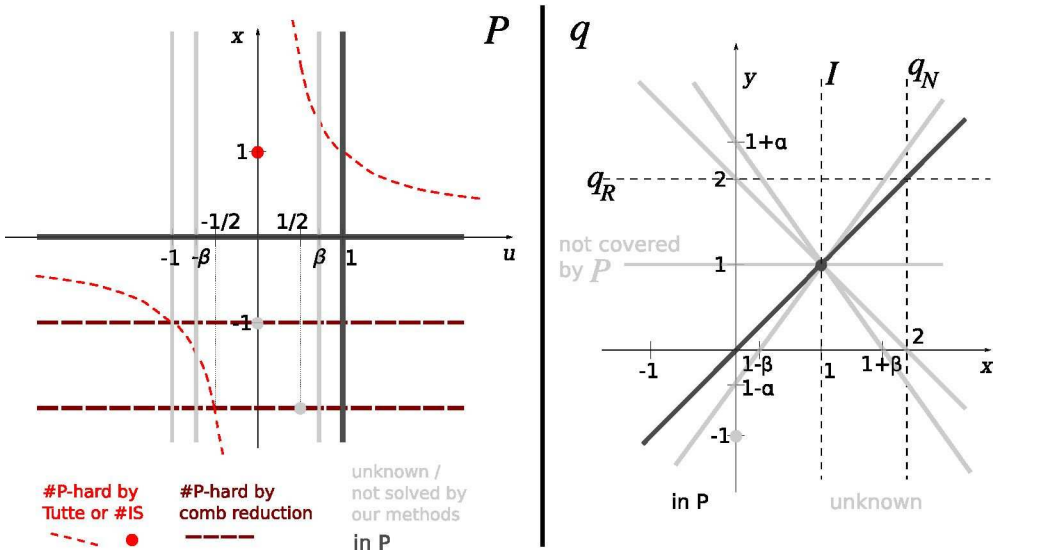


Figure 1: Complexity of the interlace polynomials P and q. $\alpha = \sqrt{2}$, $\beta = 1/\sqrt{2}$

Theorem 3.5. *Let G be a graph and G_k be obtained out of G by performing a k-comb operation at every vertex. Then*

$$P(G_k; u, x) = p(k, u, x)^{|V(G)|} P(G; u, x/p(k, u, x)), \tag{3.6}$$

where $p(k, u, x) = (1 + x)^k(xu^2 + 1) - xu^2$.

4. Complexity of evaluating the Interlace Polynomial exactly

The goal of this section is to uncover the complexity maps for P and q as indicated in Figure 1. While the left hand side (complexity map for P) is intended to follow the *arguments* which prove the hardness, the right hand side (complexity map for q) focuses on presenting the *results*.

Remark 4.1. $P(\mu, 0)$ and $P(1, \xi)$ are trivially solvable in polynomial time for any $\mu, \xi \in \tilde{\mathbb{Q}}$, as $P(G; \mu, 0) = 1$ and $P(G; 1, \xi) = (1 + \xi)^{|V|}$. ∎

Thus, on the thick black lines $x = 0$ and $u = 1$ in the left half of Figure 1, P can be evaluated in polynomial time. By Lemma 2.3, these lines in the complexity map for P correspond to the point $(1, 1)$ and the line $x = y$, resp., in the complexity map for q, see the right half of Figure 1.

4.1. Identifying hard points

We want to establish Corollary 4.4 and Remark 4.5 which tell us, that P is #P-hard to evaluate almost everywhere on the dashed hyperbola in Figure 1 and at $(0, 1)$. To this end we collect known hardness results about the interlace polynomial.

Let $t(G;x,y)$ denote the Tutte polynomial of an undirected graph $G=(V,E)$. It may be defined by its state expansion as

$$t(G;x,y)=\sum_{B\subseteq E(G)}(x-1)^{r(E)-r(B)}(y-1)^{|B|-r(B)}, \tag{4.1}$$

where $r(B)$ is the $\mathbb{F}_2$-rank of the *incidence* matrix of $G[B]=(V,B)$, the subgraph of G induced by B. (Note that $r(B)$ equals the number of vertices of $G[B]$ minus the number of components of $G[B]$, which is the rank of B in the cycle matriod of G.) For details about the Tutte polynomial we refer to standard literature [Tut84, BO92, Wel93]. The complexity of the Tutte polynomial has been studied extensively. In particular, the following result is known.

Theorem 4.2 ([Ver05])**.** *Evaluating the Tutte polynomial of planar graphs at (ξ,ξ) is $\#\mathsf{P}$-hard for all $\xi\in\tilde{\mathbb{Q}}$ except for $\xi\in\{0,1,2,1\pm\sqrt{2}\}$.*

We will profit from this by a connection between the interlace polynomial and the Tutte polynomial of planar graphs. This connection is established via medial graphs. For any planar graph G one can build the oriented medial graph $\vec{G}_m$, find an Euler circuit C in $\vec{G}_m$ and obtain the circle graph H of C. The whole procedure can be performed in polynomial time. For details we refer to [EMS06]. We will use

Theorem 4.3 ([ABS04a, End of Section 7]; [EMS06, Theorem 3.1])**.** *Let G be a planar graph, $\vec{G}_m$ be the oriented medial graph of G and H be the circle graph of some Euler circuit C of $\vec{G}_m$. Then $q(H;2,y)=t(G;y,y)$. Thus we have $t(\upsilon,\upsilon)\preceq^{\mathtt{m}} P(\frac{1}{\upsilon-1},\upsilon-1)$, where $t(\upsilon,\upsilon)$ denotes the problem of evaluating the Tutte polynomial of a* planar *graph at (υ,υ).*

Proof. See the references for $q(H;2,y)=t(G;y,y)$ and use Lemma 2.3. ∎

We set $\alpha=\sqrt{2}$ and $\beta=1/\sqrt{2}$. Let $B_1=\{\pm1,\pm\beta,0\}$. Theorem 4.2 and Theorem 4.3 yield

Corollary 4.4. *Evaluating the vertex-nullity interlace polynomial q_N is $\#\mathsf{P}$-hard almost everywhere. In particular, we have:*

- *The problem $q_N(2)$ is trivially solvable in polynomial time.*
- *For any $\upsilon\in\tilde{\mathbb{Q}}\setminus\{0,1,2,1\pm\alpha\}$ the problem $q_N(\upsilon)=q(2,\upsilon)$ is $\#\mathsf{P}$-hard. Or, in other words, for any $\mu\in\tilde{\mathbb{Q}}\setminus B_1$ the problem $P(\mu,1/\mu)$ is $\#\mathsf{P}$-hard.* ∎

Remark 4.5. $P(0,1)$ is $\#\mathsf{P}$-hard, as $P(G;0,1)$ equals the number of independent sets of G, which is $\#\mathsf{P}$-hard to compute [Val79]. ∎

4.2. Reducing to hard points

The cloning reduction allows us to spread the collected hardness over almost the whole plane: Combining Corollary 4.4 and Remark 4.5 with Proposition 3.4 we obtain

Proposition 4.6. *Let $B_1=\{\pm1,\pm\beta,0\}$ and $B_2=\{0,-1,-2\}$ (as defined on Pages 104 and 102, resp.). Let $(\mu,\xi)\in((\tilde{\mathbb{Q}}\setminus B_1)\cup\{0\})\times(\tilde{\mathbb{Q}}\setminus B_2)$. Then $P(\mu,\xi)$ is $\#\mathsf{P}$-hard.* ∎

This tells us that P is #P-hard to evaluate at every point in left half of Figure 1 not lying on one of the seven thick lines (three of which are solid gray ones, two of which are solid black ones, and two of which are dashed ones). Using the comb reduction we are able to reveal the hardness of the interlace polynomial P on the lines $x = -1$ and $x = -2$:

Proposition 4.7. *For $\mu \in \tilde{\mathbb{Q}} \setminus B_1$ the problem $P(\mu, -1)$ is #P-hard.*

Proposition 4.8. *For $\mu \in ((\tilde{\mathbb{Q}} \setminus B_1) \setminus \{\frac{1}{2}\}) \cup \{0\}$ the problem $P(\mu, -2)$ is #P-hard.*

The proofs of the preceding propositions can be found in [BH07].

4.3. Summing up

First we summarize our knowledge about P.

Theorem 4.9. *Let $\beta = 1/\sqrt{2}$.*

1. *$P(\mu, \xi)$ is computable in polynomial time on the lines $\mu = 1$ and $\xi = 0$.*
2. *For $(\mu, \xi) \in \big((\tilde{\mathbb{Q}} \setminus \{-1, -\beta, \beta, 1\}) \times (\tilde{\mathbb{Q}} \setminus \{0\})\big) \setminus \{(1/2, -2)\}$ the problem $P(\mu, \xi)$ is #P-hard.*

Proof. Summary of Remark 4.1, Proposition 4.6, Proposition 4.7, Proposition 4.8. The hardness of $P(0, -1)$ follows from Corollary 4.10. ∎

We have not given any argument why $P(0, -1)$ is #P-hard. This follows from [AM07].

Corollary 4.10. *Evaluating the independent set polynomial $I(\lambda) = P(0, \lambda) = q(1, 1+\lambda)$ is #P-hard at all $\lambda \in \tilde{\mathbb{Q}}$ except at $\lambda = 0$, where it is computable in polynomial time.*

Proof. The matching generating polynomial of a graph G is defined as $\sum_{k \geq 0} m(G; k)x^k$, where $m(G; k)$ denotes the number of matching of size k in G. [AM07] proves that $g(\xi)$ is #P-hard for all $\xi \in \mathbb{R} \setminus \{0\}$. As the matchings of a graph are the independent sets of its line graph, the result follows. ∎

Remark 4.11. Note that, except for the point $(0, -1)$, the statement of Corollary 4.10 is also a direct consequence of Proposition 4.6 and Proposition 4.8, without using [AM07]. ∎

Now we turn to the complexity of q, see also the right half of Figure 1.

Theorem 4.12. *The two-variable interlace polynomial q is #P-hard to evaluate almost everywhere. In particular, we have:*

1. *$q(\xi, \upsilon)$ is computable in polynomial time on the line $\xi = \upsilon$.*
2. *Let $\xi \in \tilde{\mathbb{Q}} \setminus \{1\}$ and x be an indeterminate. Then $q(\xi, 1)$ is as hard as computing the whole polynomial $q(x, 1)$.*
3. *$q(\xi, \upsilon)$ is #P-hard for all*

$$(\xi, \upsilon) \in \{(\xi, \upsilon) \in \tilde{\mathbb{Q}}^2 \mid \upsilon \neq \pm(\xi - 1) + 1 \text{ and } \upsilon \neq \pm\sqrt{2}(\xi - 1) + 1 \text{ and } \upsilon \neq 1 \text{ and } (\xi, \upsilon) \neq (0, -1)\}.$$

Proof of Theorem 4.12 (Sketch). (1) and (3) follow from Remark 4.1 and Theorem 4.9 using Lemma 2.3. For $\xi \neq 1$, (3.5) gives $q(G_k; \xi, 1) = q(G; k(\xi - 1) + 1, 1)$, which yields enough points for interpolation in the same way as in Proposition 3.4 using $k = 1, 2, 3, \ldots$ This proves (2). ∎

Theorem 4.12 implies

Corollary 4.13. *Let $\beta = 1/\sqrt{2}$. Evaluating the vertex-rank interlace polynomial $q_R(G; x)$ is #P-hard at all $\xi \in \tilde{\mathbb{Q}}$ except at $\xi = 0, 1 - \beta, 1 + \beta$ (complexity open) and $\xi = 2$ (computable in polynomial time).* ∎

5. Inapproximability of the Independent Set Polynomial

Provided we can evaluate the independent set polynomial at some fixed point, cloning (combing, resp.) of vertices allows us to evaluate it at very large points. In this section we exploit this to prove that the independent set polynomial is hard to approximate. Similar results are shown in [GJ07] for the Tutte polynomial.

Definition 5.1. Let $\lambda \in \tilde{\mathbb{Q}}$. By a randomized 2^{n^k}-approximation algorithm for $I(\lambda)$ we mean a randomized algorithm, that, given a graph G with n nodes, runs in time polynomial in n and returns $\tilde{I}(G; \lambda) \in \tilde{\mathbb{Q}}$ such that

$$\Pr[2^{-n^k} I(G; \lambda) \leq \tilde{I}(G; \lambda) \leq 2^{n^k} I(G; \lambda)] \geq \frac{3}{4}.$$

In [GJ07], (non)approximability in the weaker sense of (not) admitting an FPRAS is considered.

Definition 5.2. Let $\lambda \in \tilde{\mathbb{Q}}$. A fully polynomial randomized approximation scheme (FPRAS) for $I(\lambda)$ is a randomized algorithm, that given a graph G with n nodes and an error tolerance $\varepsilon, 0 < \varepsilon < 1$, runs in time polynomial in n and $1/\varepsilon$ and returns $\tilde{I}(G; \lambda) \in \tilde{\mathbb{Q}}$ such that

$$\Pr[2^{-\varepsilon} I(G; \lambda) \leq \tilde{I}(G; \lambda) \leq 2^{\varepsilon} I(G; \lambda)] \geq \frac{3}{4}.$$

Lemma 5.3. *For every $\lambda \in \tilde{\mathbb{Q}}$, $0 \neq |1 + \lambda| \neq 1$, and every $k \in \mathbb{N}$ there is no randomized polynomial time 2^{n^k}-approximation algorithm for $I(\lambda)$ unless* RP = NP.

Theorem 5.4. *For every $\lambda \in \tilde{\mathbb{Q}} \setminus \{-1, 0\}$ and every $k \in \mathbb{N}$ there is no randomized polynomial time 2^{n^k}-approximation algorithm (and thus also no FPRAS) for $I(\lambda)$ unless* RP = NP.

Proof. Lemma 5.3 gives the inapproximability at $\lambda \in \tilde{\mathbb{Q}} \setminus \{-2, -1, 0\}$. By (3.6) we could turn an approximation algorithm for $I(-2)$ into an approximation algorithm for $I(2)$ which would imply RP = NP by Lemma 5.3. ∎

Proof of Lemma 5.3. Assume we have $\lambda, 0 \neq |1+\lambda| \neq 1$, $k \in \mathbb{N}$ and a randomized 2^{n^k}-approximation algorithm $\mathcal{A}$ for $I(\lambda)$. Given a graph G, Theorem 3.3 and Theorem 3.5, resp., will allow us to evaluate the independent set polynomial at a point ξ with $|\xi|$ that large, that an approximation of $I(G;\xi)$ can be used to recover the degree of $I(G;x)$, which is the size of a maximal independent set of G. As computing this number is NP-hard, a randomized 2^{n^k}-approximation algorithm for $I(G;\lambda)$ would yield an RP-algorithm for an NP-hard problem, which implies RP = NP.

Let $G = (V, E)$ be a graph with $|V| = n$. We distinguish two cases. If $|1+\lambda| > 1$, we choose a positive integer l such that with $\xi := (1+\lambda)^l - 1$ we have

$$|\xi| \geq 2^{n^{k+1}}. \tag{5.1}$$

This can be achieved by choosing $l = \mathsf{poly}(n)$. (Let m be an integer such that $|\lambda + 1| \geq 2^{1/(n^m)}$. Then we can choose $l = (n^{k+1}+1)n^m$.) If $0 < |1+\lambda| < 1$, we choose a positive integer l such that with $\xi := \frac{\lambda}{(1+\lambda)^l}$ we have (5.1). By Theorem 3.3 (Theorem 3.5, resp.) we have $I(G;\xi) = I(G_l;\lambda)$ ($I(G;\xi) = (1+\lambda)^{-l|V|} I(G_l;\lambda)$, resp.). Algorithm $\mathcal{A}$ returns on input G_l within time $\mathsf{poly}(nl) = \mathsf{poly}(n)$ an approximation $\tilde{I}(G_l;\lambda)$, such that with $\tilde{I}(G;\xi) := \tilde{I}(G_l;\lambda)$ ($\tilde{I}(G;\xi) := \frac{\tilde{I}(G_l;\lambda)}{(1+\lambda)^{l|V|}}$, resp.) we have

$$2^{-n^k} I(G;\xi) \leq \tilde{I}(G;\xi) \leq 2^{n^k} I(G;\xi) \tag{5.2}$$

with high probability.

Let c be the size of a maximal independent set of G, and let N be the number of independent sets of maximal size. We have

$$I(G;x) = Nx^c + \sum_{0 \leq j \leq c-1} i(G;j) x^j$$

and thus

$$\begin{aligned} \Big| \frac{I(G;\xi)}{\xi^c} - N \Big| &\leq \sum_{0 \leq j \leq c-1} i(G;j) |\xi|^{j-c} \\ &\leq c 2^n |\xi|^{-1} \leq 2^{\log n + n - n^{k+1}} < \frac{1}{2} \end{aligned} \tag{5.3}$$

for large n. If we could evaluate $I(G;\xi)$ exactly, we could try all $c \in \{1, \ldots, n\}$ to find the one for which $\frac{I(G;\xi)}{\xi^c}$ is a good estimation for N, $1 \leq N \leq 2^n$. This c is unique by (5.1). The following calculation shows that this is also possible using the approximation algorithm $\mathcal{A}$.

Using $\mathcal{A}$ we compute $\tilde{N}(\tilde{c}) := \frac{\tilde{I}(G;\xi)}{\xi^{\tilde{c}}}$ for all $\tilde{c} \in \{1, \ldots, n\}$. We claim that c is the unique $\tilde{c}$ with

$$2^{-n^k - 1} \leq \tilde{N}(\tilde{c}) \leq 2^{n^k + n + 1}. \tag{5.4}$$

Let us prove this claim. As $1 \leq N \leq 2^n$ and by (5.3), we know that

$$\frac{1}{2} \leq \frac{I(G,\xi)}{\xi^c} \leq 2^{n+1}. \tag{5.5}$$

Thus, by (5.2), $\tilde{c} = c$ fulfills (5.4).

On the other hand, when $\tilde{c} \leq c-1$ we have

$$|\tilde{N}(\tilde{c})| \overset{(5.2),(5.5)}{\geq} 2^{-n^k-1}|\xi| \overset{(5.1)}{\geq} 2^{n^{k+1}-n^k-1} > 2^{n^k+n+1}$$

for large n. When $\tilde{c} \geq c+1$ we have $|\tilde{N}(\tilde{c})| < 2^{-n^k-1}$ by similar arguments. This shows that any integer $\tilde{c}, \tilde{c} \neq c$, does not fulfill (5.4). Thus, c can be found in randomized polynomial time using $\mathcal{A}$. ∎

Acknowledgement

We would like to thank Johann A. Makowsky for valuable comments on an earlier version of this work and for drawing our attention to [AM07].

References

[ABCS00] Richard Arratia, Béla Bollobás, Don Coppersmith, and Gregory B. Sorkin. Euler circuits and DNA sequencing by hybridization. *Discrete Applied Mathematics*, 104(1-3):63–96, 2000.

[ABS04a] Richard Arratia, Béla Bollobás, and Gregory B. Sorkin. The interlace polynomial of a graph. *J. Comb. Theory Ser. B*, 92(2):199–233, 2004.

[ABS04b] Richard Arratia, Béla Bollobás, and Gregory B. Sorkin. A two-variable interlace polynomial. *Combinatorica*, 24(4):567–584, 2004.

[AM07] Ilia Averbouch and J. A. Makowsky. The complexity of multivariate matching polynomials, February 2007. Preprint.

[BH07] Markus Bläser and Christian Hoffmann. On the complexity of the interlace polynomial, 2007. Preprint, `arXiv:cs.CC/0707.4565`.

[BM06] Markus Bläser and Johann Makowsky. Hip hip hooray for Sokal, 2006. Unpublished note.

[BO92] Thomas Brylawski and James Oxley. The Tutte polynomial and its applications. In Neil White, editor, *Matroid Applications*, Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1992.

[Cou07] Bruno Courcelle. A multivariate interlace polynomial, 2007. Preprint, `arXiv:cs.LO/0702016v2`.

[EMS06] Joanna A. Ellis-Monaghan and Irasema Sarmiento. Distance hereditary graphs and the interlace polynomial, 2006. Preprint, arXiv:math.CO/0604088 v2.

[GJ07] Leslie Ann Goldberg and Mark Jerrum. Inapproximability of the Tutte polynomial. In *STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 459–468, New York, NY, USA, 2007. ACM Press.

[JVW90] F. Jaeger, D. L. Vertigan, and D. J. A. Welsh. On the computational complexity of the Jones and the Tutte polynomials. *Math. Proc. Cambridge Philos. Soc.*, 108:35–53, 1990.

[Sok05] Alan D. Sokal. The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In Bridget S. Webb, editor, *Surveys in Combinatorics 2005*. Cambridge University Press, 2005.

[Tut84] W. T. Tutte. *Graph Theory*. Addison Wesley, 1984.

[Val79] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Ver05] Dirk Vertigan. The computational complexity of Tutte invariants for planar graphs. *SIAM Journal on Computing*, 35(3):690–712, 2005.

[Wel93] D. J. A. Welsh. *Complexity: knots, colourings and counting*. Cambridge University Press, New York, NY, USA, 1993.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 109-120
www.stacs-conf.org

MINIMIZING FLOW TIME IN THE WIRELESS GATHERING PROBLEM

VINCENZO BONIFACI [1,3], PETER KORTEWEG [2], ALBERTO MARCHETTI-SPACCAMELA [3], AND LEEN STOUGIE [2,4]

[1] Technische Universität Berlin, Institut für Mathematik, Berlin, Germany

[2] Eindhoven University of Technology, Dept of Mathematics and Computer Science, Eindhoven, The Netherlands
E-mail address: p.korteweg@tue.nl,l.stougie@tue.nl

[3] University of Rome "La Sapienza", Dept of Computer and Systems Science, Rome, Italy
E-mail address: bonifaci@dis.uniroma1.it,alberto@dis.uniroma1.it

[4] CWI, Amsterdam, The Netherlands
E-mail address: stougie@cwi.nl

Abstract. We address the problem of efficient data gathering in a wireless network through multi-hop communication. We focus on the objective of minimizing the maximum flow time of a data packet. We prove that no polynomial time algorithm for this problem can have approximation ratio less than $\Omega(m^{1/3})$ when m packets have to be transmitted, unless $\mathbf{P} = \mathbf{NP}$. We then use resource augmentation to assess the performance of a FIFO-like strategy. We prove that this strategy is 5-speed optimal, i.e., its cost remains within the optimal cost if we allow the algorithm to transmit data at a speed 5 times higher than that of the optimal solution we compare to.

1. Introduction

Wireless networks are used in many areas of practical interest, such as mobile phone communication, ad-hoc networks, and radio broadcasting. Moreover, recent advances in miniaturization of computing devices equipped with short range radios have given rise to strong interest in sensor networks for their relevance in many practical scenarios (environment control, accident monitoring etc.) [1, 16].

In many applications of wireless networks data gathering is a critical operation for extracting useful information from the operating environment: information collected from multiple nodes in the network should be transmitted to a sink that may process the data, or act as a gateway to other networks. We remark that in the case of wireless sensor networks

1998 ACM Subject Classification: C.2.2: Computer-Communication Networks – Network Protocols; F.2.2: Analysis of Algorithms and Problem Complexity – Nonnumerical Algorithms and Problems. General terms: Algorithms, Design, Theory.

Key words and phrases: wireless networks, data gathering, approximation algorithms, distributed algorithms.

sensor nodes have limited computation capabilities, thus implying that data gathering is an even more crucial operation. For this reasons, data gathering in sensor networks has received significant attention in the last few years; we cite just a few contributions [1, 10]. The problem finds also applications in Wi-Fi networks when many users need to access a gateway using multi-hop wireless relay-routing [5].

In this paper we focus on the problem of designing and analysing *simple distributed* algorithms that have *good approximation guarantees* in *realistic scenarios*. Namely, we are interested in algorithms that are not only distributed but that are fast and can be implemented with limited overhead: sophisticated algorithms that require solving complex combinatorial optimization problems are impractical for implementations and have mainly theoretical interest.

In order to formally assess the performance of the proposed algorithms we focus on the minimization of the maximum flow, i.e. minimizing the maximum time spent in the system by a packet. Almost all of the previous literature considered the objective of minimizing the completion time (see for example [3–5, 10, 11, 13, 17]). Flow minimization is a largely used criterion in scheduling theory that more suitably allows to assess the quality of service provided when multiple requests occur over time [7, 8, 12, 15].

The problem of modelling realistic scenarios of wireless sensor networks is complicated by the many parameters that define the communication among nodes and influence the performance of transmissions (see for example [1, 18]). In the sequel we assume that stations have a common clock, hence time can be divided into rounds. Each node is equipped with a half-duplex interface; as a result it cannot send and receive during the same round. Typically, not all nodes in the network can communicate with each other directly, hence packets have to be sent through several nodes before they can be gathered at the sink; this is called *multi-hop* routing.

The key issue in our setting is *interference*. A radio signal has a *transmission* radius, the distance over which the signal is strong enough to send data, and an *interference* radius, the distance over which the radio signal is strong enough to interfere with other radio signals. If node i is transmitting data to node j we have *interference* (or *collision*) in communication if j also receives signals from other stations at the same time. Following Bermond et al. [5], we model the wireless network using a graph and a parameter d_I. An edge between nodes i and j represents the fact that stations i and j are within transmission range of each other. The parameter d_I models the interference radius: a node j successfully receives a signal if one of his neighbors is transmitting, and no other node within distance d_I from j is transmitting in the same round. The case $d_I = 1$ has been extensively considered earlier (see e.g. [4, 10, 11]); but we remark that assuming $d_I = 1$ or assuming that interferences/transmissions are modeled according to the well known unit disk graph model *does not* adequately represent interferences as they occur in practice [18].

Kumar et al. [14] give an overview of other interference models, including the so-called *distance-2 interference model*. The distance-2 interference model is similar to our interference model with $d_I = 1$, plus the extra constraint that no two transmitting nodes should be adjacent; we observe that this requirement might pose unnecessary conditions.

The Wireless Gathering Problem. An instance of the *Wireless Gathering Problem* (WGP) is given by a static wireless network which consists of several stations (nodes) and one base station (the sink), modeled as a graph, together with the interference radius d_I; over time data packets arrive at stations that have to be gathered at the base station.

A feasible solution of an instance of WGP is a schedule without interference which determines for each packet both route and times at which it is sent. As we will see in Section 2 this can be modeled as a clean combinatorial optimization problem. We consider two different objectives. One is to minimize completion time, i.e., the time needed to gather all the packets. Another, perhaps more natural, objective is minimization of maximum flow time of packets, i.e., the maximum difference between release time and arrival time at the sink of a packet. We call these two problems C-WGP and F-WGP, respectively.

Related work. The Wireless Gathering Problem was introduced by Bermond et al. [5] in the context of wireless access to the Internet in villages. The authors proved that the problem of minimizing the completion time is **NP**-hard and presented a greedy algorithm with asymptotic approximation ratio at most 4. In [6] we considered arbitrary release times and proposed an on-line greedy algorithm with the same approximation ratio.

Bar-Yehuda et al. [4] considered distributed algorithms for C-WGP. Their model is a special case of our model, where $d_I = 1$ and there are no release dates. Kumar et al. [13] studied the more general *end-to-end* transmission problem, where each of the packets may have a different source *and* destination in the network. Under the assumption of a distance-2 interference model, Kumar et al. considered the objective of minimizing the maximum completion time of the schedule, and presented hardness results and approximation algorithms for arbitrary graphs and disk graphs. They presented distributed algorithms for packet scheduling over fixed routing paths, and used a linear program in order to determine the paths. By contrast, we use a shortest paths tree to fix the routing paths, which is easier to implement in a distributed setting.

Florens et al. [10] considered the minimization of the completion time of data gathering in a setting with unidirectional antennas. They provided a 2-approximation algorithm for tree networks and an optimal algorithm for line networks. Gargano and Rescigno [11] gave a polynomial time algorithm for the special case of the same model in which each node has exactly one packet to send.

Another related problem is to compute the throughput of a wireless network. This has been studied for example in [14]. We also observe that many papers study broadcasting in wireless networks [3, 17]. However, we stress that data gathering and broadcasting are substantially different tasks in the context of packet networks. In particular, the idea of reversing a broadcast schedule to obtain a gathering schedule (which works when data can be aggregated) cannot be used.

Main results. In Section 3 we give inapproximability results for F-WGP. We prove that F-WGP on m packets cannot be approximated within $\Omega(m^{1/3})$, unless $\mathbf{P} = \mathbf{NP}$. We also show that any algorithm using shortest paths in order to route the packets to the sink is no better than an $\Omega(m)$-approximation.

In Section 4 we present a polynomial time resource augmented approximation algorithm for F-WGP which is in fact an on-line algorithm. We use resource augmentation because F-WGP is hard to approximate within a reasonable factor.

Resource augmentation was introduced in the context of machine scheduling in [12]: the idea is to study the performance of on-line algorithms which are given processors faster than the adversary. Intuitively, this has been done to compensate an on-line scheduler for its lack of future information. Such an approach has led to a number of interesting results showing that moderately faster processors are sufficient to attain satisfactory performance guarantee for various scheduling problems, e.g. [8, 12]

Surprisingly, in the case of F-WGP a modest resource augmentation allows to compensate not only the lack of future information but also the approximation hardness of the problem: we present a σ-speed optimal algorithm for F-WGP and C-WGP; σ depends on d_I and is at most 5.

We remark that our algorithm can be implemented using local information only: in particular, it suffices that a node is informed about the state of nodes within distance $d_I + 1$. On the other hand, our lower bounds hold for centralized algorithms as well.

2. Mathematical preliminaries

We formulate WGP as a graph optimization problem. The model we use can be seen as a generalization of a well-known model for packet radio networks [3, 4].

An instance of WGP consists of a graph $G = (V, E)$, a *sink* node $s \in V$, a positive integer d_I, and a set of *data packets* $J = \{1, 2, \ldots, m\}$. Each packet $j \in J$ has an *origin* $o_j \in V$ and a *release date* $r_j \in \mathbb{Z}_+$.

We assume that time is discrete; we call a time unit a *round*. The rounds are numbered $0, 1, 2, \ldots$. During each round a node may either be *sending* a packet, be *receiving* a packet or be *inactive*. If two nodes u and v are adjacent, then u can send a packet to v during a round. If node u sends a packet j to v in some round, then the pair (u, v) is said to be a *call* from u to v. For each pair of nodes $u, v \in V$, the *distance* between u and v, denoted by $d(u, v)$, is the length of a shortest path from u to v in G. Two calls (u, v) and (u', v') *interfere* if they occur in the same round and either $d(u', v) \leq d_I$ or $d(u, v') \leq d_I$; otherwise the calls are *compatible*. For this reason, the parameter d_I is called the *interference radius*. The special case of a unit interference radius corresponds to the above cited model of Bar-Yehuda et al. [3].

For every packet $j \in J$, the release date r_j specifies the time at which the packet enters the network, i.e. packet j cannot be sent before round r_j. In the off-line version the entire instance is completely known at time 0; in the on-line version information about a packet becomes known only at its release date.

A solution for a WGP instance is a schedule of compatible calls such that all packets are ultimately sent to the sink. Notice that while in principle each radio transmission can broadcast the same packet to multiple destinations, in the gathering problem having more than one copy of the same packet does not help, as it suffices to keep the one that will arrive first at the sink. Thus, we assume that at any time there is a unique copy of each packet. Also, in the model we consider, packets cannot be aggregated.

Given a schedule, let x_j^t be the unique node holding packet j at time t. The integer $C_j := \min\{t : x_j^t = s\}$ is called the *completion time* of packet j, while $F_j := C_j - r_j$ is the *flow time* of packet j. In this paper we are interested in the minimization of $\max_j F_j$ (F-WGP). As an intermediate step in the analysis of F-WGP, we also study the minimization of $\max_j C_j$ (C-WGP).

Some auxiliary notation, we denote by $\delta_j := d(o_j, s)$ the minimum number of calls required for packet j to reach s. We also define $\gamma := d_I + 2$, and $\gamma_0 := \lfloor (d_I + 1)/2 \rfloor$.

We analyze the performance of our algorithms using the standard worst case analysis techniques of approximation ratio analysis, as well as resource augmentation. Given a WGP instance $\mathcal{I}$ and an algorithm ALG, we define $\mathcal{C}(\mathcal{I})$ as the cost of ALG and $\mathcal{C}^*(\mathcal{I})$ as the cost of the optimal solution on $\mathcal{I}$. A polynomial-time algorithm is called an α-approximation if for any instance $\mathcal{I}$ we have $\mathcal{C}(\mathcal{I}) \leq \alpha \cdot \mathcal{C}^*(\mathcal{I})$.

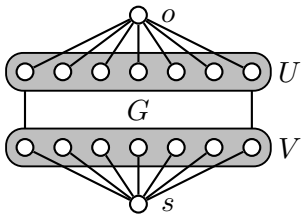


Figure 1: The construction in the proof of Theorem 3.2

In the resource augmentation paradigm, the algorithm is allowed to use more resources than the adversary. We consider resource augmentation based on speed, in which we assume that the algorithm can schedule compatible calls with higher speed than the optimal algorithm. For any $\sigma \geq 1$, we call an algorithm a σ-speed algorithm if the time used by the algorithm to schedule a set of compatible calls is $1/\sigma$ time units. See [2] for more information on approximation algorithms, and [12] for more on resource augmentation.

3. Inapproximability

In this section we prove an inapproximability result for F-Wgp. To prove this result we consider the so-called *induced matching* problem. A matching M in a graph G is an *induced matching* if no two edges in M are joined by an edge of G. The following rather straightforward relation between compatible calls in a bipartite graph and induced matchings will be crucial in the following.

Proposition 3.1. *Let $G = (U, V, E)$ be a bipartite graph with node sets (U, V) and edge set E. Then, a set $M \subseteq E$ is an induced matching if and only if the calls corresponding to edges of M, directed from U to V, are all pairwise compatible, assuming $d_I = 1$.* ∎

Induced Bipartite Matching (IBM)
Instance: a bipartite graph G and an integer k.
Question: does G have an induced matching of size at least k?

We will use the fact that the optimization version of IBM is hard to approximate: there exists an $\alpha > 1$ such that it is **NP**-hard to distinguish between graphs with induced matchings of size k and graphs in which all induced matchings are of size at most k/α. The current best bound for α is 6600/6599 [9].

Theorem 3.2. *Unless* $\mathbf{P} = \mathbf{NP}$, *no polynomial-time algorithm can approximate* F-Wgp *within a ratio better than* $\Omega(m^{1/3})$.

Proof. Let (G, k) be an instance of IBM, $G = (U, V, E)$. We construct a 4-layer network with a unique source o (first layer), a clique on U and a clique on V (middle layers), and a sink s (last layer). Source o is adjacent to each node in U, and s to each node in V. The edges between U and V are the same as in G (see Figure 1). We set $d_I = 1$.

The F-Wgp instance consists of $m := (1 - 1/\alpha)^{-1}(1 + k/\alpha)(2k + 1)k = \Theta(k^3)$ packets with origin o. They are divided into m/k groups of size k. Each packet in the hth group has release date $(k + 1)h$, $h = 0, \ldots, m/k - 1$. Rounds $(k + 1)h$ till $(k + 1)(h + 1) - 1$ together are a *phase*.

We prove that if G has an induced matching of size k, there is a gathering schedule of cost $2k + 1$, while if G has no induced matching of size more than k/α, every schedule has cost at least $(2k + 1)k = (2k + 1)\Theta(m^{1/3})$. The theorem then follows directly.

Assume G has an induced matching M of size k, say (u_i, v_i), $i = 0 \dots k-1$. Then consider the following gathering schedule. In each phase, the k new packets at o are transmitted, necessarily one-by-one, to layer U while old packets at layer V (if any) are absorbed at the sink; then, in a *single* round, the k new packets move from U to V via the matching edges. More precisely, each phase can be scheduled in $k+1$ rounds as follows:
1. for $i = 0, \dots, k-1$ execute in the ith round the two calls (o, u_i) and $(v_{i+1 \bmod k}, s)$;
2. in the kth round, execute simultaneously all the calls (u_i, v_i), $i = 0, \dots, k-1$.
The maximum flow time of the schedule is $2k+1$, as a packet released in phase h reaches the sink before the end of phase $h+1$.

In the other direction, assume that each induced matching of G is of size at most k/α. By Proposition 3.1, at most k/α calls can be scheduled in any round from layer U to layer V. We ignore potential interference between calls from o to U and calls from V to s; doing so may only decrease the cost of a schedule. As a consequence, we can assume that each packet follows a shortest path from o to s. Notice however that, due to the cliques on the layers U and V, no call from U to V is compatible with a call from o to U, or with a call from V to s.

Let m_o and m_U be the number of packets at o and U, respectively, at the beginning of a given phase. Also, let $\beta := 1 + k/\alpha$. We associate to the phase a potential value $\psi := \beta m_o + m_U$, and we show that at the end of the phase the potential will have increased proportionally to k. Let c_o and c_U denote the number of calls from o to U and from U to V, respectively, during the phase. Since a phase consists of $k+1$ rounds, and in each round at most k/α calls are scheduled from U to V, we have $c_o + c_U/(k/\alpha) \le k+1$, or, equivalently since $k/\alpha = \beta - 1$,

$$(\beta - 1)c_o + c_U \le (\beta - 1)(k+1). \tag{3.1}$$

If m'_o, m'_U are the number of packets at o and U at the beginning of the next phase, and $\psi' = \beta m'_o + m'_U$ is the new potential, we have

$$\begin{aligned}
m'_o &= m_o + k - c_o \\
m'_U &= m_U + c_o - c_U \\
\psi' - \psi &= \beta(m'_o - m_o) + m'_U - m_U \\
&= \beta(k - c_o) + c_o - c_U \\
&= \beta k - (\beta - 1)c_o - c_U \\
&\ge \beta k - (\beta - 1)(k+1) \\
&= k - (\beta - 1) \\
&= (1 - 1/\alpha)k
\end{aligned}$$

where the inequality uses (3.1).

Thus, consider the situation after m/k phases. The potential has become at least $\Psi := (1 - 1/\alpha)m$. By definition of the potential, this implies that at least $\Psi/\beta = (1 - 1/\alpha)(1+k/\alpha)^{-1}m = (2k+1)k$ packets reside at either o or U; in particular, they have been released but not yet absorbed at the sink. Since the sink cannot receive more than one packet per round, this clearly implies a maximum flow time of $(2k+1)k = (2k+1)\Theta(m^{1/3})$ for one of these packets. ∎

In cases where the packets are routed via shortest paths to the sink – a behavior common to many gathering protocols – the result of Theorem 3.2 can be strengthened further.

Theorem 3.3. *No algorithm that routes packets along shortest paths can approximate* F-WGP *within a ratio better than* $\Omega(m)$.

Proof. Consider the instance in Figure 2. The adversary releases a message at each of the nodes u_1, u_2, u_3 at times $5i$, $i = 0, \ldots, m/3$. Any shortest paths following algorithm sends all messages via u, yielding $\max_j C_j \geq 3m$. As $r_j \leq 5m/3$ for each message j, we have $\max_j F_j \geq 3m - 5m/3 = 4m/3$.

The adversary sends each message over the path which does not contain u. We claim that it is possible to do this so that all messages released at time $5i$ arrive at the sink in round $5(i+1)+1$ latest. If the claim holds, then we have $\max_j F_j^* \leq 5(i+1)+1-5i = 6$, from which the theorem will follow.

We prove the claim by induction. Suppose the claim holds for messages released in round $5(i-1)$. Then, the last message released at time $5(i-1)$ latest is sent to the sink in round $5i$. This message does not block any message released in round $5i$. Now, the adversary sends the messages released in round $5i$ to a node adjacent to s in 3 rounds, i.e. in the rounds $5i$, $5i+1$ and $5i+2$. Then, it requires another 3 rounds to send all 3 messages to the sink, i.e. the rounds $5i+3$, $5i+4$, and $5(i+1)$. This proves the theorem, since $\max_j F_j / \max_j F_j^* \geq (4m/3)/6 = 2m/9$. ∎

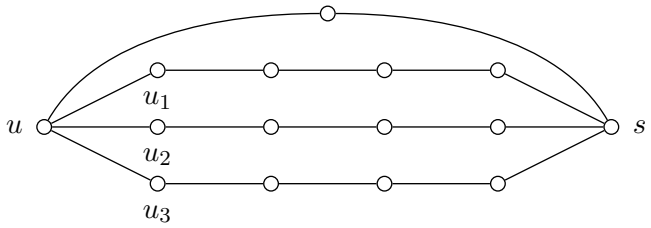


Figure 2: No shortest path based algorithm is better than $\Omega(m)$-approximate ($d_I = 1$).

4. Approximation Algorithms

In this section we present and analyze a FIFO algorithm for WGP. First, we show that FIFO is a 5-approximation for C-WGP. Note that the best approximation algorithm known is 4-approximate; the main interest in analyzing FIFO is that we use it as a subroutine in an algorithm for F-WGP which uses resource augmentation. Next, we prove that this algorithm with resource augmentation is a σ-speed optimal algorithm, for any $\sigma \geq 5$, for both C-WGP and F-WGP.

4.1. An approximation algorithm for C-WGP

We will present an approximation algorithm for C-WGP. The algorithm we consider is actually a special case of a general scheme for which we can prove an upper bound on the completion time [6]. In this scheme, called PRIORITY GREEDY, each packet is assigned a unique priority based on some algorithm-specific rules. Then, in each round, packets are considered in order of decreasing priority and are sent towards the sink as long as there is no interference with higher priority packets.

Algorithm 4.1 (PRIORITY GREEDY)**.** In every round, consider the available packets in order of decreasing priority, and send each next packet along a shortest path from its current node to s, as long as this causes no interference with any higher-priority packet.

We first derive upper bounds on the completion time C_j of each packet j in a PRIORITY GREEDY solution.

We say that packet j is *blocked* in round t if $t \geq r_j$ but j is not sent in round t. Note that in a PRIORITY GREEDY algorithm a packet can only be blocked due to interference with a higher priority packet. We define the following *blocking relation* on a PRIORITY GREEDY schedule: $k \prec j$ if in the last round in which j is blocked, k is the packet closest to j that is sent in that round and has a priority higher than j (ties broken arbitrarily). The blocking relation induces a directed graph $F = (J, A)$ on the packet set J with an arc (k, j) for each $k, j \in J$ such that $k \prec j$. Observe that, for any PRIORITY GREEDY schedule, F is a directed forest and the root of each tree of F is a packet which is never blocked. For each j let $T(j) \subseteq F$ be the tree of F containing j, $b(j) \in J$ be the root of $T(j)$, and $P(j)$ the set of packets along the path in F from $b(j)$ to j. Finally, define $\pi_j := \min\{\delta_j, \gamma_0\}$ and $R_j := r_j + \delta_j - \pi_j$.

We have upper and lower bounds on the completion time of a packet.

Lemma 4.2 ([6]). *For each packet $j \in J$, $C_j \leq R_{b(j)} + (\gamma/\gamma_0) \cdot \sum_{i \in P(j)} \pi_i$.*

Lemma 4.3 ([6]). *Let $S \subseteq J$ be a nonempty set of packets, and let C_i^* denote the completion time of packet i in some feasible schedule. Then there is $k \in S$ such that $\max_{i \in S} C_i^* \geq R_k + \sum_{i \in S} \pi_i$.*

Our algorithm is based on a version of the PRIORITY GREEDY scheme, in which a higher priority is given to packets with earlier release dates (ties broken arbitrarily). We call this algorithm FIFO after the famous *first-in-first-out* algorithm in scheduling and service systems, though in our case packets do not necessarily arrive in order of their priority at the sink.

Theorem 4.4. FIFO *is a $(1 + \gamma/\gamma_0)$-approximation algorithm for* C-WGP.

Proof. Let j be the packet having maximum C_j, and consider $T(j)$, the tree containing j in the forest induced by the blocking relation. We can apply Lemma 4.3 with $S = T(j)$ to obtain

$$\max_{i \in T(j)} C_i^* \geq r_k + \delta_k + \sum_{\substack{i \in T(j) \\ i \neq k}} \pi_i \tag{4.1}$$

where k is some packet in $T(j)$. On the other hand, by using Lemma 4.2,

$$\begin{aligned} C_j &\leq R_{b(j)} + \frac{\gamma}{\gamma_0} \sum_{i \in P(j)} \pi_i \\ &= r_{b(j)} + \delta_{b(j)} - \pi_{b(j)} + \frac{\gamma}{\gamma_0} \sum_{i \in P(j)} \pi_i \\ &\leq r_{b(j)} + \frac{\gamma}{\gamma_0} \min\{\delta_k, \gamma_0\} + \frac{\gamma}{\gamma_0} \sum_{\substack{i \in P(j) \\ i \neq k}} \pi_i + \delta_{b(j)} \\ &\leq \frac{\gamma}{\gamma_0} \left(r_k + \delta_k + \sum_{\substack{i \in T(j) \\ i \neq k}} \pi_i \right) + \delta_{b(j)}. \end{aligned} \tag{4.2}$$

where we used the fact that, by definition of FIFO, we have $r_{b(j)} \leq r_k$. Equations (4.1) and (4.2), and observation $\max_{i \in T(j)} C_i^* \geq \delta_{b(j)}$ prove the theorem. ∎

It is straightforward to verify that $2 \leq \gamma/\gamma_0 \leq 4$ for all d_I, while $\gamma/\gamma_0 = 3$ for $d_I = 1$.

Corollary 4.5. FIFO *is a* 5*-approximation algorithm for* C-WGP. *When* $d_I = 1$, FIFO *is a* 4*-approximation for* C-WGP.

The bound on the approximation ratio of FIFO is slightly worse than that of a PRIORITY GREEDY algorithm based on R_j, which is a γ/γ_0-approximation. In fact, we also have an example on which FIFO is strictly worse than a γ/γ_0-approximation (we omit the example here due to space limitations). However, we remark that FIFO is both natural and simple; and, perhaps more importantly, Theorem 4.4 will be instrumental in proving good bounds for the minimization of maximum flow time, where we will use FIFO as a subroutine of our algorithm.

4.2. A resource augmentation bound for F-WGP

Motivated by the hardness result of Section 3, we study algorithms under resource augmentation. In this context we study σ-speed algorithms, in which data packets are sent at a speed that is σ times faster than the solution we compare to.

Algorithm 4.6 (σ-FIFO)**.**
1. Create a new instance $\mathcal{I}'$ by multiplying release dates: $r'_j := \sigma r_j$;
2. Run FIFO on $\mathcal{I}'$;
3. Speed up the schedule thus obtained by a factor of σ.

The schedule constructed by σ-FIFO is a feasible σ-speed solution to the original problem because of step 1. We will show that σ-FIFO is optimal for both C-WGP and F-WGP, if $\sigma \geq \gamma/\gamma_0 + 1$. The following Lemma is crucial.

Lemma 4.7. *If* σ-FIFO *is a* σ*-speed optimal algorithm for* C-WGP*, then it is also a* σ*-speed optimal algorithm for* F-WGP.

Proof. Let F^*_j and $F_{j,\sigma}$ be the flow time of data packet j in an optimal solution and in a σ-FIFO solution, respectively, to F-WGP and let C^*_j and $C_{j,\sigma}$ be the completion time of data packet j in the same solutions. Suppose σ-FIFO is a σ-speed optimal algorithm for C-WGP, hence we have $\max_{j\in J} C_{j,\sigma} \leq \max_{j\in J} C^*_j$. We show that this inequality implies, for any time t,

$$\max_{j\in J,\, r_j = t} C_{j,\sigma} \leq \max_{j\in J,\, r_j \leq t} C^*_j. \tag{4.3}$$

We prove inequality (4.3) by contradiction. Suppose it is false, then there is an instance $\mathcal{I}$ of minimum size (number of data packets) for which it is false. Also, let t_0 be the first round in such an instance for which it is false. By definition, σ-FIFO schedules each data packet j definitively in round r_j; no data packet is rescheduled in a later round. I.e., the algorithm determines the completion time $C_{j,\sigma}$ of data packet j in round r_j. If the inequality is false, then we must have

$$C_{i,\sigma} > \max_{j\in J,\, r_j \leq t_0} C^*_j, \tag{4.4}$$

for some data packet i with $r_i = t_0$, and because $\mathcal{I}$ is a minimum size instance the instance does not contain any data packets released after round t_0. But then (4.4) contradicts

$\max_{j\in J} C_{j,\sigma} \le \max_{j\in J} C^*_j$. Using (4.3) we have

$$\begin{aligned}\max_{j\in J} F_{j,\sigma} &= \max_t \left(\max_{j\in J,\, r_j = t} C_{j,\sigma} - t \right) \le \max_t \left(\max_{j\in J,\, r_j \le t} C^*_j - t \right) \\ &\le \max_t \left(\max_{j\in J,\, r_j \le t} F^*_j \right) = \max_{j\in J} F^*_j .\end{aligned}$$

■

Theorem 4.8. *For $\sigma \ge \gamma/\gamma_0 + 1$, σ-FIFO is a σ-speed optimal algorithm for both* C-WGP *and* F-WGP.

Proof. By Lemma 4.7, it suffices to prove that σ-FIFO is σ-speed optimal for C-WGP.

Let C_j be the completion time of any data packet j in the σ-FIFO solution on instance $\mathcal{I}$, and let C'_j be the completion time of j in the FIFO solution on the instance $\mathcal{I}'$ (see the algorithm description). By construction $C_j = C'_j/\sigma$. Let $R'_j := \sigma r_j + \delta_j - \pi_j$. Then the upper bound of Lemma 4.2 applied to instance $\mathcal{I}'$ implies $C'_j \le R'_{b(j)} + (\sigma - 1)\sum_{i\in P(j)} \pi_i$. Hence,

$$C_j = C'_j/\sigma \le \frac{1}{\sigma} R'_{b(j)} + \frac{\sigma - 1}{\sigma} \sum_{i\in P(j)} \pi_i \;\le\; r_{b(j)} + \frac{1}{\sigma}\delta_{b(j)} + \frac{\sigma - 1}{\sigma} \sum_{i\in P(j)} \pi_i . \tag{4.5}$$

Since in any solution $b(j)$ has to reach the sink we clearly have

$$\max_{i\in P(j)} C^*_i \ge C^*_{b(j)} \ge r_{b(j)} + \delta_{b(j)} . \tag{4.6}$$

Also, by Lemma 4.3, for some $k \in P(j)$,

$$\max_{i\in P(j)} C^*_i \ge R_k + \sum_{i\in P(j)} \pi_i \ge r_k + \sum_{i\in P(j)} \pi_i \ge r_{b(j)} + \sum_{i\in P(j)} \pi_i , \tag{4.7}$$

where the last inequality follows from $b(j)$ having lowest release time in $P(j)$, by definition of FIFO. Combining (4.5), (4.6) and (4.7), we obtain

$$\begin{aligned}\max_{i\in P(j)} C^*_i &= \frac{1}{\sigma} \max_{i\in P(j)} C^*_i + \frac{\sigma - 1}{\sigma} \max_{i\in P(j)} C^*_i \\ &\ge \frac{1}{\sigma}\left(r_{b(j)} + \delta_{b(j)} \right) + \frac{\sigma - 1}{\sigma}\left(r_{b(j)} + \sum_{i\in P(j)} \pi_i \right) \\ &= r_{b(j)} + \frac{1}{\sigma}\delta_{b(j)} + \frac{\sigma - 1}{\sigma} \sum_{i\in P(j)} \pi_i \ge C_j .\end{aligned}$$

■

Corollary 4.9. 5-FIFO *is a* 5*-speed optimal algorithm for* C-WGP *and* F-WGP.

4.3. Another upper bound for FIFO

As we have seen in Section 3, F-WGP is extremely hard to approximate without resource augmentation – no bound better than $\Omega(m^{1/3})$ is possible. Moreover, algorithms that route along shortest paths cannot do better than $\Omega(m)$ (recall Theorem 3.3). In this section we show that FIFO is in fact an $O(m)$-approximation for F-WGP. Thus, apart from constant factors, FIFO is best possible among algorithms that use shortest paths.

Theorem 4.10. FIFO *is an* $O(m)$*-approximation for* F-WGP.

Proof. Since every packet must be gathered at the sink, clearly $\max_j F_j^* \geq \max_j \delta_j \geq \max_j \pi_j$. Now let j be the packet incurring the maximum flow time in the schedule obtained by FIFO. Since $r_j \geq r_{b(j)}$ (by definition of FIFO), we have

$$R_{b(j)} - r_j = r_{b(j)} + \delta_{b(j)} - \pi_{b(j)} - r_j \leq \delta_{b(j)} \tag{4.8}$$

Using Lemma 4.2 and (4.8), we get

$$\begin{aligned} F_j = C_j - r_j &\leq R_{b(j)} - r_j + \frac{\gamma}{\gamma_0} \sum_{i \in P(j)} \pi_i \\ &\leq \delta_{b(j)} + \frac{\gamma}{\gamma_0} \sum_{i \in P(j)} \pi_i \\ &\leq \max_i F_i^* + \frac{\gamma}{\gamma_0} \cdot |P(j)| \cdot \max_i F_i^* \\ &\leq \left(1 + \frac{\gamma}{\gamma_0} m\right) \max_i F_i^*. \end{aligned}$$

■

5. Conclusion

We considered the wireless gathering problem with the objective of minimizing the maximum flow time of data packets (F-WGP). We showed that the simple on-line algorithm FIFO has favorable behavior: although the problem is extremely hard to approximate in general, augmenting the transmission rate by a factor of 5 allows FIFO to remain within the cost of an optimal solution for the problem without augmentation.

It is an open question whether optimality can be achieved by augmenting the transmission rate by a factor smaller than 5, and whether an efficient algorithm exists that matches the $\Omega(m^{1/3})$ lower bound on the approximability of F-WGP.

Another interesting set of questions concerns resource augmentation by allowing the algorithms to use extra frequencies, meaning that more than one data packet can be sent simultaneously over the same channel. For instance, does there exist a 5-frequency optimal FIFO-type algorithm?

For the minimization of the completion time (C-WGP), the existence of a polynomial time approximation scheme is still open. It is known that no algorithm that uses shortest paths to route the data packets to the sink can give an improvement over the currently best approximation ratio [6]. It is a challenge to design and analyze congestion avoiding algorithms with better ratios.

Acknowledgments

Research supported by EU FET Integrated Project AEOLUS IST-15964, by FET EC 6th FP Research Project ARRIVAL FP6-021235-2, by the Dutch BSIK-BRICKS project, and by MIUR-FIRB Italy-Israel project RBIN047MH9.

References

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

[2] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, 1999.

[3] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in multihop radio networks: an exponential gap between determinism and randomization. *Journal of Computer and Systems Sciences*, 45(1):104–126, 1992.

[4] R. Bar-Yehuda, A. Israeli, and A. Itai. Multiple communication in multihop radio networks. *SIAM Journal on Computing*, 22(4):875–887, 1993.

[5] J. Bermond, J. Galtier, R. Klasing, N. Morales, and S. Pérennes. Hardness and approximation of gathering in static radio networks. *Parallel Processing Letters*, 16(2):165–183, 2006.

[6] V. Bonifaci, P. Korteweg, A. Marchetti-Spaccamela, and L. Stougie. An approximation algorithm for the wireless gathering problem. In *Proc. 10th Scandinavian Workshop on Algorithm Theory*, pages 328–338, 2006.

[7] H.-L. Chan, T. W. Lam, and K.-S. Liu. Extra unit-speed machines are almost as powerful as speedy machines for competitive flow time scheduling. In *Proc. 17th Symp. on Discrete Algorithms*, pages 334–343, 2006.

[8] C. Chekuri, A. Goel, S. Khanna, and A. Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *Proc. 36th Symp. on Theory of Computing*, pages 363–372, 2004.

[9] W. Duckworth, D. Manlove, and M. Zito. On the approximability of the maximum induced matching problem. *Journal of Discrete Algorithms*, 3(1):79–91, 2005.

[10] C. Florens, M. Franceschetti, and R. J. McEliece. Lower bounds on data collection time in sensory networks. *IEEE Journal on Selected Areas in Communications*, 22:1110– 1120, 2004.

[11] L. Gargano and A. A. Rescigno. Optimally fast data gathering in sensor networks. In *Proc. 31st Symp. on Mathematical Foundations of Computer Science*, pages 399–411, 2006.

[12] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[13] V. S. Anil Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. End-to-end packet-scheduling in wireless ad-hoc networks. In J. I. Munro, editor, *Proc. 15th Symp. on Discrete Algorithms*, pages 1021–1030, 2004.

[14] V. S. Anil Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. Algorithmic aspects of capacity in wireless networks. In *Measurement and Modeling of Computer Systems*, pages 133–144, 2005.

[15] J. McCullough and E. Torng. SRPT optimally utilizes faster machines to minimize flow time. In J. I. Munro, editor, *Proc. 15th Symp. on Discrete Algorithms*, pages 350–358, 2004.

[16] K. Pahlavan and A. H. Levesque. *Wireless information networks*. Wiley, New York, 1995.

[17] A. Pelc. Broadcasting in radio networks. In *Handbook of Wireless Networks and Mobile Computing*, pages 509–528. Wiley and Sons, 2002.

[18] S. Schmid and R. Wattenhofer. Algorithmic models for sensor networks. In *Proc. 20th Int. Parallel and Distributed Processing Symposium*, 2006.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 121-132
www.stacs-conf.org

ON TERMINATION FOR FAULTY CHANNEL MACHINES

PATRICIA BOUYER [1], NICOLAS MARKEY [1], JOËL OUAKNINE [2], PHILIPPE SCHNOEBELEN [1], AND JAMES WORRELL [2]

[1] LSV, ENS Cachan, CNRS
61 Av. Pdt. Wilson, F-94230 Cachan, France
{bouyer,markey,phs}@lsv.ens-cachan.fr

[2] Oxford University Computing Laboratory
Wolfson Bldg., Parks Road, Oxford OX1 3QD, UK
{joel,jbw}@comlab.ox.ac.uk

Abstract. A *channel machine* consists of a finite controller together with several fifo channels; the controller can read messages from the head of a channel and write messages to the tail of a channel. In this paper, we focus on channel machines with *insertion errors*, i.e., machines in whose channels messages can spontaneously appear. Such devices have been previously introduced in the study of Metric Temporal Logic. We consider the *termination* problem: are all the computations of a given insertion channel machine finite? We show that this problem has non-elementary, yet primitive recursive complexity.

1. Introduction

Many of the recent developments in the area of automated verification, both theoretical and practical, have focussed on infinite-state systems. Although such systems are not, in general, amenable to fully algorithmic analysis, a number of important classes of models with decidable problems have been identified. Several of these classes, such as Petri nets, process algebras, process rewrite systems, faulty channel machines, timed automata, and many more, are instances of *well-structured transition systems*, for which various problems are decidable—see [7] for a comprehensive survey.

Well-structured transition systems are predicated on the existence of 'compatible well-quasi orders', which guarantee, for example, that certain fixed-point computations will terminate, etc. Unfortunately, these properties are often non-constructive in nature, so that although convergence is guaranteed, the rate of convergence is not necessarily known. As a result, the computational complexity of problems involving well-structured transition systems often remains open.

Key words and phrases: Automated Verification, Computational Complexity.

Thanks: Patricia Bouyer is also affiliated with the Oxford University Computing Laboratory and is partially supported by a Marie Curie Fellowship.

In this paper, we are interested in a particular kind of well-structured transition systems, known as faulty channel machines. A channel machine (also known as a queue automaton) consists of a finite-state controller equipped with several unbounded fifo channels (queues, buffers). Transitions of the machine can write messages (letters) to the tail of a channel and read messages from the head of a channel. Channel machines can be used, for example, to model distributed protocols that communicate asynchronously.

Channel machines, unfortunately, are easily seen to be Turing powerful [3], and all non-trivial verification problems concerning them are therefore undecidable. In [1, 6, 4, 2], Abdulla and Jonsson, and Finkel *et al.* independently introduced *lossy channel machines* as channel machines operating over an unreliable medium; more precisely, they made the assumption that messages held in channels could at any point vanish nondeterministically. Not only was this a compelling modelling assumption, more adequately enabling the representation of fault-tolerant protocols, for example, but it also endowed the underlying transition systems of lossy channel machines with a well-structure, thanks to Higman's lemma [8]. As a result, several non-trivial problems, such as control-state reachability, are decidable for lossy channel machines.

Abdulla and Jonsson admitted in [1] that they were unable to determine the complexity of the various problems they had shown to be decidable. Such questions remained open for almost a decade, despite considerable research interest in the subject from the scientific community. Finally, Schnoebelen showed in [16] that virtually all non-trivial decidable problems concerning lossy channel machines have non-primitive recursive complexity. This result, in turn, settled the complexity of a host of other problems, usually via reduction from reachability for lossy channel machines. Recently, the relevance of the lossy channel model was further understood when it was linked to a surprisingly complex variant of Post's correspondence problem [5].

Other models of unreliable media in the context of channel machines have also been studied in the literature. In [4], for example, the effects of various combinations of insertion, duplication, and lossiness errors are systematically examined. Although insertion errors are well-motivated (as former users of modems over telephone lines can attest!), they were surprisingly found in [4] to be theoretically uninteresting: channels become redundant, since read- and write-transitions are continuously enabled (the former because of potential insertion errors, the latter by assumption, as channels are unbounded). Consequently, most verification problems trivially reduce to questions on finite automata.

Recently, however, slightly more powerful models of channel machines with insertion errors have appeared as key tools in the study of Metric Temporal Logic (MTL). In [13, 14], the authors showed that MTL formulas can capture the computations of insertion channel machines *equipped with primitive operations for testing channel emptiness.* This new class of faulty channel machines was in turn shown to have a non-primitive recursive reachability problem and an undecidable recurrent control-state reachability problem. Consequently, MTL satisfiability and model checking were established to be non-primitive recursive over finite words [13], and undecidable over infinite words [14].

Independently of Metric Temporal Logic, the notion of emptiness testing, broadly construed, is a rather old and natural one. Counter machines, for instance, are usually assumed to incorporate primitive zero-testing operations on counters, and likewise pushdown automata are able to detect empty stacks. Variants of Petri nets have also explored emptiness testing for places, usually resulting in a great leap in computational power. In the context of channel machines, a slight refinement of emptiness testing is *occurrence testing*, checking

that a given channel contains no occurrence of a particular message, as defined and studied in [14]. Emptiness and occurrence testing provide some measure of control over insertion errors, since once a message has been inserted into a channel, it remains there until it is read off it.

Our main focus in this paper is the complexity of the *termination* problem for insertion channel machines: given such a machine, are all of its computations finite? We show that termination is non-elementary, yet primitive recursive. This result is quite surprising, as the closely related problems of reachability and recurrent reachability are respectively non-primitive recursive and undecidable. Moreover, the mere *decidability* of termination for insertion channel machines follows from the theory of well-structured transition systems, in a manner quite similar to that for lossy channel machines. In the latter case, however, termination is non-primitive recursive, as shown in [16]. Obtaining a primitive recursive upper bound for insertion channel machines has therefore required us to abandon the well-structure and pursue an entirely new approach.

On the practical side, one of the main motivations for studying termination of insertion channel machines arises from the safety fragment of Metric Temporal Logic. Safety MTL was shown to be decidable in [15], although no non-trivial bounds on the complexity could be established at the time. It is not difficult, however, to show that (non-)termination for insertion channel machines reduces (in polynomial time) to satisfiability for Safety MTL; the latter, therefore, is also non-elementary. We note that in a similar vein, a lower bound for the complexity of satisfiability of an extension of Linear Temporal Logic was given in [10], via a reduction from the termination problem for counter machines with incrementation errors.

2. Decision Problems for Faulty Channel Machines: A Brief Survey

In this section, we briefly review some key decision problems for lossy and insertion channel machines (the latter equipped with either emptiness or occurrence testing). Apart from the results on termination and structural termination for insertion channel machines, which are presented in the following sections, all results that appear here are either known or follow easily from known facts. Our presentation is therefore breezy and terse. Background material on well-structured transition systems can be found in [7].

The *reachability* problem asks whether a given distinguished control state of a channel machine is reachable. This problem was shown to be non-primitive recursive for lossy channel machines in [16]; it is likewise non-primitive recursive for insertion channel machines via a straightforward reduction from the latter [13].

The *termination* problem asks whether all computations of a channel machine are finite, starting from the initial control state and empty channel contents. This problem was shown to be non-primitive recursive for lossy channel machines in [16]. For insertion channel machines, we prove that termination is non-elementary in Section 4 and primitive recursive in Section 5.

The *structural termination* problem asks whether all computations of a channel machine are finite, starting from the initial control state but regardless of the initial channel contents. This problem was shown to be undecidable for lossy channel machines in [12]. For insertion channel machines, it is easy to see that termination and structural termination coincide, so that the latter is also non-elementary primitive-recursive decidable.

	Lossy Channel Machines	**Insertion Channel Machines**
Reachability	non-primitive recursive	non-primitive recursive
Termination	non-primitive recursive	non-elementary / primitive recursive
Struct. term.	undecidable	non-elementary / primitive recursive
Response	undecidable	non-primitive recursive
Recurrence	undecidable	undecidable
CTL / LTL	undecidable	undecidable

Figure 1: Complexity of decision problems for faulty channel machines.

Given a channel machine $\mathcal{S}$ and two distinguished control states p and q of $\mathcal{S}$, a *response* property is an assertion that every p state is always eventually followed by a q state in any infinite computation of $\mathcal{S}$. Note that a counterexample to a response property is a computation that eventually visits p and forever avoids q afterwards. The undecidability of response properties for lossy channel machines follows easily from that of structural termination, as the reader may wish to verify.

In the case of insertion channel machines, response properties are decidable, albeit at non-primitive recursive cost (by reduction from reachability). For decidability one first shows using the theory of well-structured transition systems that the set of all reachable configurations, the set of p-configurations, and the set of configurations that have infinite q-avoiding computations are all effectively computable. It then suffices to check whether their mutual intersection is empty.

The *recurrence* problem asks, given a channel machine and a distinguished control state, whether the machine has a computation that visits the distinguished state infinitely often. It is undecidable for lossy channel machines by reduction from response, and was shown to be undecidable for insertion channel machines in [14].

Finally, *CTL and LTL model checking* for both lossy and insertion channel machines are undecidable, which can be established along the same lines as the undecidability of recurrence.

These results are summarised in Figure 1.

3. Definitions

A *channel machine* is a tuple $\mathcal{S} = (Q, init, \Sigma, C, \Delta)$, where Q is a finite set of control states, $init \in Q$ is the initial control state, Σ is a finite channel alphabet, C is a finite set of channel names, and $\Delta \subseteq Q \times L \times Q$ is the transition relation, where $L = \{c!a, c?a, c{=}\emptyset, a{\notin}c : c \in C, a \in \Sigma\}$ is the set of transition labels. Intuitively, label $c!a$ denotes the writing of message a to tail of channel c, label $c?a$ denotes the reading of message a from the head of channel c, label $c{=}\emptyset$ tests channel c for emptiness, and label $a{\notin}c$ tests channel c for the absence (non-occurrence) of message a.

We first define an *error-free* operational semantics for channel machines. Given $\mathcal{S}$ as above, a *configuration* of $\mathcal{S}$ is a pair (q, U), where $q \in Q$ is the control state and $U \in (\Sigma^*)^C$ gives the contents of each channel. Let us write *Conf* for the set of possible configurations of $\mathcal{S}$. The rules in Δ induce an L-labelled transition relation on *Conf*, as follows:

(1) $(q, c!a, q') \in \Delta$ yields a transition $(q, U) \xrightarrow{c!a} (q', U')$, where $U'(c) = U(c){\cdot}a$ and $U'(d) = U(d)$ for $d \neq c$. *In other words, the channel machine moves from control*

state q to control state q', writing message a to the tail of channel c and leaving all other channels unchanged.

(2) $(q, c?a, q') \in \Delta$ yields a transition $(q, U) \xrightarrow{c?a} (q', U')$, where $U(c) = a \cdot U'(c)$ and $U'(d) = U(d)$ for $d \neq c$. *In other words, the channel machine reads message a from the head of channel c while moving from control state q to control state q', leaving all other channels unchanged.*

(3) $(q, c{=}\emptyset, q') \in \Delta$ yields a transition $(q, U) \xrightarrow{c=\emptyset} (q', U)$, provided $U(c)$ is the empty word. *In other words, the transition is only enabled if channel c is empty; all channel contents remain the same.*

(4) $(q, a{\notin}c, q') \in \Delta$ yields a transition $(q, U) \xrightarrow{a\notin c} (q', U)$, provided a does not occur in $U(c)$. *In other words, the transition is only enabled if channel c contains no occurrence of message a; all channels remain unchanged.*

If the only transitions allowed are those listed above, then we call $\mathcal{S}$ an *error-free* channel machine. This machine model is easily seen to be Turing powerful [3]. As discussed earlier, however, we are interested in channel machines with (potential) *insertion errors*; intuitively, such errors are modelled by postulating that channels may at any time acquire additional messages interspersed throughout their current contents.

For our purposes, it is convenient to adopt the *lazy* model of insertion errors, given next. Slightly different models, such as those of [4, 14], have also appeared in the literature. As the reader may easily check, all these models are equivalent insofar as reachability and termination properties are concerned.

The lazy operational semantics for channel machines with insertion errors simply augments the transition relation on *Conf* with the following rule:

(5) $(q, c?a, q') \in \Delta$ yields a transition $(q, U) \xrightarrow{c?a} (q', U)$. *In other words, insertion errors occur 'just in time', immediately prior to a read operation; all channel contents remain unchanged.*

The channel machines defined above are called *insertion channel machines with occurrence testing*, or *ICMOT*s. We will also consider *insertion channel machines with emptiness testing*, or *ICMET*s. The latter are simply ICMOTs without any occurrence-testing transitions (i.e., transitions labelled with $a{\notin}c$).

A *run* of an insertion channel machine is a finite or infinite sequence of transitions of the form $\sigma_0 \xrightarrow{l_0} \sigma_1 \xrightarrow{l_1} \ldots$ that is consistent with the lazy operational semantics. The run is said to start from the initial configuration if the first control state is *init* and all channels are initially empty.

Our main focus in this paper is the study of the complexity of the *termination* problem: given an insertion channel machine $\mathcal{S}$, are all runs of $\mathcal{S}$ starting from the initial configuration finite?

4. Termination is Non-Elementary

In this section, we show that the termination problem for insertion channel machines—ICMETs and ICMOTs—is non-elementary. More precisely, we show that the termination problem for ICMETs of size n in the worst case requires time at least $2{\Uparrow}\Omega(\log n)$.[1] Note that the same immediately follows for ICMOTs.

[1]The expression $2{\Uparrow}m$, known as tetration, denotes an exponential tower of 2s of height m.

Our proof proceeds by reduction from the termination problem for two-counter machines in which the counters are tetrationally bounded; the result then follows from standard facts in complexity theory (see, e.g., [9]).

Without insertion errors, it is clear that a channel machine can directly simulate a two-counter machine simply by storing the values of the counters on one of its channels. To simulate a counter machine in the presence of insertion errors, however, we require periodic integrity checks to ensure that the representation of the counter values has not been corrupted. Below we give a simulation that follows the 'yardstick' construction of Meyer and Stockmeyer [17, 11]: roughly speaking, we use an m-bounded counter to check the integrity of a 2^m-bounded counter.

Theorem 4.1. *The termination problem for ICMETs and ICMOTs is non-elementary.*

Proof. Let us say that a counter is m-bounded if it can take values in $\{0, 1, \ldots, m-1\}$. We assume that such a counter u comes equipped with procedures $\textsc{Inc}(u)$, $\textsc{Dec}(u)$, $\textsc{Reset}(u)$, and $\textsc{IsZero}(u)$, where Inc and Dec operate modulo m, and increment, resp. decrement, the counter. We show how to simulate a deterministic counter machine $\mathcal{M}$ of size n equipped with two $2{\Uparrow}n$-bounded counters by an ICMET $\mathcal{S}$ of size $2^{O(n)}$. We use this simulation to reduce the termination problem for $\mathcal{M}$ to the termination problem for $\mathcal{S}$.

By induction, assume that we have constructed an ICMET $\mathcal{S}_k$ that can simulate the operations of a $2{\Uparrow}k$-bounded counter u_k. We assume that $\mathcal{S}_k$ correctly implements the operations $\textsc{Inc}(u_k)$, $\textsc{Dec}(u_k)$, $\textsc{Reset}(u_k)$, and $\textsc{IsZero}(u_k)$ (in particular, we assume that the simulation of these operations by $\mathcal{S}_k$ is guaranteed to terminate). We describe an ICMET $\mathcal{S}_{k+1}$ that implements a $2{\Uparrow}(k+1)$-bounded counter u_{k+1}. $\mathcal{S}_{k+1}$ incorporates $\mathcal{S}_k$, and thus can use the above-mentioned operations on the counter u_k as subroutines. In addition, $\mathcal{S}_{k+1}$ has two extra channels c and d on which the value of counter u_{k+1} is stored in binary. We give a high-level description.

We say that a configuration of $\mathcal{S}_{k+1}$ is *clean* if channel c has size $2{\Uparrow}k$ and channel d is empty. We ensure that all procedures on counter u_{k+1} operate correctly when they are invoked in clean configurations of $\mathcal{S}_{k+1}$, and that they also yield clean configurations upon completion. In fact, we only give details for the procedure $\textsc{Inc}(u_{k+1})$—see Figure 2; the others should be clear from this example.

Since the counter u_k is assumed to work correctly, the above procedure is guaranteed to terminate, having produced the correct result, in the absence of any insertion errors on channels c or d. On the other hand, insertion errors on either of these channels will be detected by one of the two emptiness tests, either immediately or in the next procedure to act on them.

The initialisation of the induction is handled using an ICMET $\mathcal{S}_1$ with no channel (in other words, a finite automaton) of size 2, which can simulate a 2-bounded counter (i.e., a single bit). The finite control of the counter machine, likewise, is duplicated using a further channel-less ICMET.

Using a product construction, it is straightforward to conflate these various ICMETs into a single one, $\mathcal{S}$, of size exponential in n (more precisely: of size $2^{O(n)}$). As the reader can easily check, $\mathcal{M}$ has an infinite computation iff $\mathcal{S}$ has an infinite run. The result follows immediately. ∎

Procedure INC(u_{k+1})
  RESET(u_k)
  repeat
    $c?x$; $d!(1-x)$ /* Increment counter u_{k+1} while transferring c to d */
    INC(u_k)
  until ISZERO(u_k) or $x = 0$
  while not ISZERO(u_k) **do**
    $c?x$; $d!x$ /* Transfer remainder of c to d */
    INC(u_k)
  endwhile
  test(c=$\emptyset$) /* Check that there were no insertion errors on c, otherwise halt */
  repeat
    $d?x$; $c!x$ /* Transfer d back to c */
    INC(u_k)
  until ISZERO(u_k)
  test(d=$\emptyset$) /* Check that there were no insertion errors on d, otherwise halt */
  return

Figure 2: Procedure to increment counter u_{k+1}. Initially, this procedure assumes that counter u_{k+1} is encoded in binary on channel c, with least significant bit at the head of the channel; moreover, c is assumed to comprise exactly $2\Uparrow k$ bits (using padding 0s if need be). In addition, channel d is assumed to be initially empty. Upon exiting, channel c will contain the incremented value of counter u_{k+1} (modulo $2\Uparrow(k+1)$) in binary, again using $2\Uparrow k$ bits, and channel d will be empty. We regularly check that no insertion errors have occurred on channels c or d by making sure that they contain precisely the right number of bits. This is achieved using counter u_k (which can count up to $2\Uparrow k$ and is assumed to work correctly) together with emptiness tests on c and d. If an insertion error does occur during execution, the procedure will either halt, or the next procedure to handle channels c and d (i.e., any command related to counter u_{k+1}) will halt.

5. Termination is Primitive Recursive

The central result of our paper is the following:

Theorem 5.1. *The termination problem for ICMOTs and ICMETs is primitive recursive. More precisely, when restricting to the class of ICMOTs or ICMETs that have at most k channels, the termination problem is in $(k+1)$-EXPSPACE.*

Proof. In what follows, we sketch the proof for ICMOTs, ICMETs being a special case of ICMOTs. Let us also assume that our ICMOTs do not make use of any emptiness tests; this restriction is harmless since any emptiness test can always be replaced by a sequence of occurrence tests, one for each letter of the alphabet, while preserving termination.

Let $\mathcal{S} = (Q, init, \Sigma, C, \Delta)$ be a fixed ICMOT without emptiness tests; in other words, $\mathcal{S}$'s set of transition labels is $L = \{c!a, c?a, a\not\in c : c \in C, a \in \Sigma\}$. Our strategy is as follows: we suppose that $\mathcal{S}$ has no infinite runs, and then derive an upper bound on the length of the longest possible finite run. The result follows by noting that the total number of possible runs is exponentially bounded by this maximal length.

For a subset $D \subseteq C$ of channels, we define an equivalence $\equiv_D$ over the set *Conf* of configurations of $\mathcal{S}$ as follows:

$$(q, U) \equiv_D (q', U') \text{ iff } q = q' \text{ and } U(d) = U'(d) \text{ for every } d \in D.$$

Let us write $Conf_D$ to denote the set $Conf/\equiv_D$ of equivalence classes of *Conf* with respect to $\equiv_D$. Furthermore, given $f : D \to \mathbb{N}$ a 'bounding function' for the channels in D, let

$$Conf^f_D = \{[(q, U)]_D \in Conf_D : |U(d)| \leq f(d) \text{ for every } d \in D\}$$

be the subset of $Conf_D$ consisting of those equivalence classes of configurations whose D-channels are bounded by f. As the reader can easily verify, we have the following bound on the cardinality γ^f_D of $Conf^f_D$:

$$\gamma^f_D \leq |Q| \prod_{d \in D} (|\Sigma| + 1)^{f(d)}. \tag{5.1}$$

Consider a finite run $\sigma_0 \xrightarrow{l_0} \sigma_1 \xrightarrow{l_1} \dots \xrightarrow{l_{n-1}} \sigma_n$ of $\mathcal{S}$ (with $n \geq 1$), where each $\sigma_i \in Conf$ is a configuration and each $l_i \in L$ is a transition label. We will occasionally write $\sigma_0 \xRightarrow{\lambda} \sigma_n$ to denote such a run, where $\lambda = l_0 l_1 \dots l_{n-1} \in L^+$.

We first state a pumping lemma of sorts, whose straightforward proof is left to the reader:

Lemma 5.2. *Let $D \subseteq C$ be given, and assume that $\sigma \xRightarrow{\lambda} \sigma'$ (with $\lambda \in L^+$) is a run of $\mathcal{S}$ such that $\sigma \equiv_D \sigma'$. Suppose further that, for every label $a \notin c$ occurring in λ, either $c \in D$, or the label $c!a$ does not occur in λ. Then λ is repeatedly firable from σ, i.e., there exists an infinite run $\sigma \xRightarrow{\lambda} \sigma' \xRightarrow{\lambda} \sigma'' \xRightarrow{\lambda} \dots$.*

Note that the validity of Lemma 5.2 rests crucially on (the potential for) insertion errors.

Let $\langle w_i \rangle_{1 \leq i \leq n}$ be a finite sequence, and let $0 < \alpha \leq 1$ be a real number. A set S is said to be α-frequent in the sequence $\langle w_i \rangle$ if the set $\{i : w_i \in S\}$ has cardinality at least αn.

The next result we need is a technical lemma guaranteeing a certain density of repeated elements in an α-frequent sequence:

Lemma 5.3. *Let $\langle w_i \rangle_{1 \leq i \leq n}$ be a finite sequence, and assume that S is a finite α-frequent set in $\langle w_i \rangle$. Then there exists a sequence of pairs of indices $\langle (i_j, i'_j) \rangle_{1 \leq j \leq \frac{\alpha n}{2(|S|+1)}}$ such that, for all $j < \frac{\alpha n}{2(|S|+1)}$, we have $i_j < i'_j < i_{j+1}$, $i'_j - i_j \leq \frac{2(|S|+1)}{\alpha}$, and $w_{i_j} = w_{i'_j} \in S$.*

Proof. By assumption, $\langle w_i \rangle$ has a subsequence of length at least αn consisting exclusively of elements of S. This subsequence, in turn, contains at least $\frac{\alpha n}{|S|+1}$ disjoint 'blocks' of length $|S| + 1$. By the pigeonhole principle, each of these blocks contains at least two identical elements from S, yielding a sequence of pairs of indices $\langle (i_j, i'_j) \rangle_{1 \leq j \leq \frac{\alpha n}{|S|+1}}$ having all the required properties apart, possibly, from the requirement that $i'_j - i_j \leq \frac{2(|S|+1)}{\alpha}$. Note also that there are, for now, twice as many pairs as required.

Consider therefore the half of those pairs whose difference is smallest, and let p be the largest such difference. Since the other half of pairs in the sequence $\langle (i_j, i'_j) \rangle$ have difference at least p, and since there is no overlap between indices, we have $\frac{1}{2} \cdot \frac{\alpha n}{|S|+1} \cdot p < n$, from which we immediately derive that p is bounded by $\frac{2(|S|+1)}{\alpha}$, as required. This concludes the proof of Lemma 5.3. ∎

Recall our assumption that $\mathcal{S}$ has no infinite run, and let $\pi = \sigma_0 \xrightarrow{l_0} \sigma_1 \xrightarrow{l_1} \ldots \xrightarrow{l_{n-1}} \sigma_n$ be any finite run of $\mathcal{S}$, starting from the initial configuration; we seek to obtain an upper bound on n.

Given a set $D \subseteq C$ of channels, it will be convenient to consider the sequence $[\pi]_D = \langle [\sigma_i]_D \rangle_{0 \leq i \leq n}$ of equivalence classes of configurations in π modulo $\equiv_D$ (ignoring the interspersed labelled transitions for now).

Let $f : C \to \mathbb{N}$ and $0 < \alpha \leq 1$ be given, and suppose that $Conf_C^f$ is α-frequent in $[\pi]_C$, so that there are at least αn occurrences of configuration equivalence classes in $Conf_C^f$ along $[\pi]_C$. Recall that $Conf_C^f$ contains γ_C^f elements. Observe, by Lemma 5.2, that no member of $Conf_C^f$ can occur twice along $[\pi]_D$, otherwise $\mathcal{S}$ would have an infinite run. Consequently,

$$n \leq \frac{\gamma_C^f}{\alpha}. \tag{5.2}$$

We will now inductively build an increasing sequence $\emptyset = D_0 \subset D_1 \subset \ldots \subset D_{|C|} = C$, as well as functions $f_i : D_i \to \mathbb{N}$ and real numbers $0 < \alpha_i \leq 1$, for $0 \leq i \leq |C|$, such that $Conf_{D_i}^{f_i}$ is α_i-frequent in $[\pi]_{D_i}$ for every $i \leq |C|$.

The base case is straightforward: the set $Conf_\emptyset^{f_0} = Conf_\emptyset$ is clearly 1-frequent in $[\pi]_\emptyset$.

Let us therefore assume that $Conf_D^f$ is α-frequent in $[\pi]_D$ for some strict subset D of C and some $f : D \to \mathbb{N}$ and $\alpha > 0$. We now compute $D' \subseteq C$ strictly containing D, $f' : D' \to \mathbb{N}$, and $\alpha' > 0$ such that $Conf_{D'}^{f'}$ is α'-frequent in $[\pi]_{D'}$.

Thanks to our induction hypothesis and Lemma 5.3, we obtain a sequence of pairs of configurations $\langle (\theta_j, \theta'_j) \rangle_{1 \leq j \leq h}$, where $h = \frac{\alpha n}{2(\gamma_D^f + 1)}$, $[\theta_j]_D = [\theta'_j]_D \in Conf_D^f$, and such that

$$\pi = \sigma_0 \Longrightarrow \theta_1 \overset{\lambda_1}{\Longrightarrow} \theta'_1 \Longrightarrow \theta_2 \overset{\lambda_2}{\Longrightarrow} \theta'_2 \Longrightarrow \ldots \Longrightarrow \theta_h \overset{\lambda_h}{\Longrightarrow} \theta'_h \Longrightarrow \sigma_n$$

with each $\lambda_j \in L^+$ having length no greater than $\frac{2(\gamma_D^f + 1)}{\alpha}$, for $1 \leq j \leq h$.

For each λ_j, let OT_j be the set of occurrence-test labels that occur at least once in λ_j. Among these sets, let OT denote the one that appears most often. Note that there are $2^{|\Sigma| \cdot |C|}$ different possible sets of occurrence-test labels, and therefore at least $\frac{h}{2^{|\Sigma| \cdot |C|}}$ of the OT_j are equal to OT.

Following a line of reasoning entirely similar to that used in Lemma 5.3[2], we can deduce that π contains at least $\frac{h}{4 \cdot 2^{|\Sigma| \cdot |C|}} = \frac{\alpha n}{8(\gamma_D^f + 1) 2^{|\Sigma| \cdot |C|}}$ non-overlapping patterns of the form

$$\theta \overset{\lambda}{\Longrightarrow} \theta' \overset{\delta}{\Longrightarrow} \bar{\theta} \overset{\bar{\lambda}}{\Longrightarrow} \bar{\theta}',$$

where:

- $[\theta]_D = [\theta']_D \in Conf_D^f$ and $[\bar{\theta}]_D = [\bar{\theta}']_D \in Conf_D^f$,
- $\lambda, \bar{\lambda} \in L^+$ each have length no greater than $\frac{2(\gamma_D^f + 1)}{\alpha}$,
- $\delta \in L^+$ has length no greater than $\frac{8(\gamma_D^f + 1) 2^{|\Sigma| \cdot |C|}}{\alpha}$, and
- the set of occurrence-test labels occurring in λ and $\bar{\lambda}$ in both cases is OT.

[2]Formally, we could directly invoke Lemma 5.3, as follows. Write the sequence of transition labels of π as $\delta_0 \lambda_1 \delta_1 \lambda_2 \cdots \lambda_h \delta_h$, with the λ_i as above. Next, formally replace each instance of λ_i whose set of occurrence-test labels is OT by a new symbol $\mathbf{O}$; if needed, add dummy non-$\mathbf{O}$ symbols to the end of the sequence to bring its length up to n, and call the resulting sequence $\langle w_i \rangle$. Finally, note that the singleton set $\{\mathbf{O}\}$ is $\frac{h}{2^{|\Sigma| \cdot |C| \cdot n}}$-frequent in $\langle w_i \rangle$.

Consider such a pattern. Observe that λ must contain at least one occurrence-test label $a{\notin}c$ with $c \notin D$ and such that the label $c!a$ occurs in λ, otherwise $\mathcal{S}$ would have an infinite run according to Lemma 5.2. Pick any such occurrence-test label and let us denote it $a{\notin}c$.

We now aim to bound the size of channel c in the $\bar{\theta}$ configuration of our patterns. Note that since λ and $\bar{\lambda}$ contain the same set of occurrence-test labels, the label $a{\notin}c$ occurs in $\bar{\lambda}$. That is to say, somewhere between configurations $\bar{\theta}$ and $\bar{\theta}'$, we know that channel c did not contain any occurrence of a. On the other hand, an a was written to the tail of channel c at some point between configurations θ and θ', since λ contains the label $c!a$. For that a to be subsequently read off the channel, the whole contents of channel c must have been read from the time of the $c!a$ transition in λ to the time of the $a{\notin}c$ transition in $\bar{\lambda}$. Finally, note that, according to our lazy operational semantics, the size of a channel changes by at most 1 with each transition. It follows that the size of channel c in configuration $\bar{\theta}$ is at most $|\lambda| + |\delta| + |\lambda'| \leq \frac{(\gamma_D^f+1)(4+8\cdot 2^{|\Sigma|\cdot|C|})}{\alpha}$.

Let $D' = D \cup \{c\}$, and define the bounding function $f' : D' \to \mathbb{N}$ such that $f'(d) = f(d)$ for all $d \in D$, and $f'(c) = \frac{(\gamma_D^f+1)(4+8\cdot 2^{|\Sigma|\cdot|C|})}{\alpha}$. From our lower bound on the number of special patterns, we conclude that the set $Conf_{D'}^{f'}$ is α'-frequent in $[\pi]_{D'}$, where $\alpha' = \frac{\alpha}{8(\gamma_D^f+1)2^{|\Sigma|\cdot|C|}}$.

We now string everything together to obtain a bound on n, the length of our original arbitrary run π. For convenience, let $c_1, c_2, \ldots, c_{|C|}$ be an enumeration of the channel names in C in the order in which they are picked in the course of our proof; thus $D_i = D_{i-1} \cup \{c_i\}$ for $1 \leq i \leq |C|$. Correspondingly, let $M_i = f_i(c_i)$, for $0 \leq i \leq |C|$, with the convention that $M_0 = 1$; it is easy to see that M_i is the maximum value of f_i over D_i, since the sequences $\langle \gamma_{D_i}^{f_i} \rangle$ and $\langle \alpha_i \rangle$ are monotonically increasing and decreasing respectively.

From Equation 5.1, we easily get that $\gamma_{D_i}^{f_i} \in O(|\mathcal{S}|^{|\mathcal{S}|M_i})$, where $|\mathcal{S}|$ is any reasonable measure of the size of our ICMOT $\mathcal{S}$. Combining this with our expressions for f' and α' above, we obtain that $M_{i+1}, \frac{1}{\alpha_{i+1}} \in O\left(\frac{|\mathcal{S}|^{|\mathcal{S}|^2 M_i}}{\alpha_i}\right)$ for $0 \leq i \leq |C| - 1$. This, in turns, lets us derive bounds for $\gamma_C^{f_{|C|}}$ and $\alpha_{|C|}$, which imply, together with Equation 5.2, that

$$n \leq 2^{2^{\cdot^{\cdot^{\cdot^{2^{P(|\mathcal{S}|)}}}}}},$$

where P is some polynomial (independent of $\mathcal{S}$), and the total height of the tower of exponentials is $|C| + 2$.

The ICMOT $\mathcal{S}$ therefore has an infinite run iff it has a run whose length exceeds the above bound. Since the lazy operational semantics is finitely branching (bounded, in fact, by the size of the transition relation), this can clearly be determined in $(|C|+1)$-EXPSPACE, which concludes the proof of Theorem 5.1. ∎

Theorems 4.1 and 5.1 immediately entail the following:

Corollary 5.4. *The structural termination problem—are all computations of the machine finite, starting from the initial control state but regardless of the initial channel contents?—is decidable for ICMETs and ICMOTs, with non-elementary but primitive-recursive complexity.*

6. Conclusion

The main result of this paper is that termination for insertion channel machines with emptiness or occurrence testing has non-elementary, yet primitive recursive complexity. This result is in sharp contrast with the equivalent problem for lossy channel machines, which has non-primitive recursive complexity.

We remark that the set of configurations from which a given insertion channel machine has at least one infinite computation is finitely representable (thanks to the theory of well-structured transition systems), and is in fact computable as the greatest fixed point of the pre-image operator. The proof of Theorem 5.1, moreover, shows that this fixed point will be reached in primitive-recursively many steps. The set of configurations from which there is an infinite computation is therefore primitive-recursively computable, in contrast with lossy channel machines for which it is not even recursive (as can be seen from the undecidability of structural termination).

Finally, another interesting difference with lossy channel machines can be highlighted by quoting a slogan from [16]: "*Lossy systems with k channels can be [polynomially] encoded into lossy systems with one channel.*" We can deduce from Theorems 4.1 and 5.1 that any such encoding, in the case of insertion channels machines, would require non-elementary resources to compute, if it were to preserve termination properties.

References

[1] Parosh Aziz Abdulla and Bengt Jonsson. Verifying programs with unreliable channels. In *Proc. 8th Annual Symposium on Logic in Computer Science (LICS'93)*, pages 160–170. IEEE Computer Society Press, 1993.

[2] Parosh Aziz Abdulla and Bengt Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.

[3] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[4] Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996.

[5] Pierre Chambart and Philippe Schnoebelen. Post embedding problem is not primitive recursive, with applications to channel systems. In *Proc. 27th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'07)*, volume 4855 of *Lecture Notes in Computer Science*, pages 265–276. Springer, 2007.

[6] Alain Finkel. Decidability of the termination problem for completely specificied protocols. *Distributed Computing*, 7(3):129–135, 1994.

[7] Alain Finkel and Philippe Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[8] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 2:326–336, 1952.

[9] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[10] Ranko Lazić. Safely freezing LTL. In *Proc. 26th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 2006.

[11] Ranko Lazić, Thomas C. Newcomb, Joël Ouaknine, A. W. Roscoe, and James Worrell. Nets with tokens which carry data. In *Proc. 28th International Conference on Application and Theory of Petri Nets (ICATPN'07)*, volume 4546 of *Lecture Notes in Computer Science*, pages 301–320. Springer, 2007.

[12] Richard Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1):35–65, 2003.

[13] Joël Ouaknine and James Worrell. On the decidability of Metric Temporal Logic. In *Proc. 19th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 188–197. IEEE Computer Society Press, 2005.

[14] Joël Ouaknine and James Worrell. On metric temporal logic and faulty Turing machines. In *Proc. 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 217–230. Springer, 2006.

[15] Joël Ouaknine and James Worrell. Safety metric temporal logic is fully decidable. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2006.

[16] Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.

[17] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proc. 5th AMS Symposium on Theory of Computing*, 1973.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 133-142
www.stacs-conf.org

STACKELBERG NETWORK PRICING GAMES

PATRICK BRIEST [1], MARTIN HOEFER [2], AND PIOTR KRYSTA [1]

[1] Department of Computer Science, The University of Liverpool, United Kingdom.
Supported by DFG grant Kr 2332/1-2 within Emmy Noether program.
E-mail address: {patrick.briest,p.krysta}@liverpool.ac.uk

[2] Department of Computer Science, RWTH Aachen University, Germany.
Supported by DFG Graduiertenkolleg 1298 "AlgoSyn".
E-mail address: mhoefer@informatik.rwth-aachen.de

Abstract. We study a multi-player one-round game termed Stackelberg Network Pricing Game, in which a *leader* can set prices for a subset of m priceable edges in a graph. The other edges have a fixed cost. Based on the leader's decision one or more *followers* optimize a polynomial-time solvable combinatorial minimization problem and choose a minimum cost solution satisfying their requirements based on the fixed costs and the leader's prices. The leader receives as revenue the total amount of prices paid by the followers for priceable edges in their solutions, and the problem is to find revenue maximizing prices. Our model extends several known pricing problems, including single-minded and unit-demand pricing, as well as Stackelberg pricing for certain follower problems like shortest path or minimum spanning tree. Our first main result is a tight analysis of a single-price algorithm for the single follower game, which provides a $(1+\varepsilon)\log m$-approximation for any $\varepsilon > 0$. This can be extended to provide a $(1+\varepsilon)(\log k + \log m)$-approximation for the general problem and k followers. The latter result is essentially best possible, as the problem is shown to be hard to approximate within $\mathcal{O}(\log^{\varepsilon} k + \log^{\varepsilon} m)$. If followers have demands, the single-price algorithm provides a $(1+\varepsilon)m^2$-approximation, and the problem is hard to approximate within $\mathcal{O}(m^{\varepsilon})$ for some $\varepsilon > 0$. Our second main result is a polynomial time algorithm for revenue maximization in the special case of Stackelberg bipartite vertex cover, which is based on non-trivial max-flow and LP-duality techniques. Our results can be extended to provide constant-factor approximations for any constant number of followers.

1. Introduction

Algorithmic pricing problems model the task of assigning revenue maximizing prices to a retailer's set of products given some estimate of the potential customers' preferences in purely computational [14], as well as strategic [3] settings. Previous work in this area has mostly focused on settings in which these preferences are rather restricted, in the sense that products are either *pure complements* [2, 7, 15, 16] and every customer is interested in exactly one subset of products or *pure substitutes* [1, 8, 10, 14, 15, 16], in which case each

1998 ACM Subject Classification: F.2 Analysis of Algorithms and Problem Complexity.
Key words and phrases: Stackelberg Games, Algorithmic Pricing, Approximation Algorithms, Inapproximability.

customer seeks to buy only a single product out of some set of alternatives. A customer's real preferences, however, are often significantly more complicated than that and therefore pose some additional challenges.

The modelling of consumer preferences has received considerable attention in the context of *algorithmic mechanism design* [18] and *combinatorial auctions* [12]. The established models range from relatively simple bidding languages to bidders that are represented by oracles allowing certain types of queries, e.g., revealing the desired bundle of items given some fixed set of prices. The latter would be a somewhat problematic assumption in the theory of pricing algorithms, where we usually assume to have access to a rather large number of potential customers through some sort of sampling procedure and, thus, are interested in preferences that allow for a compact kind of representation.

In this paper we focus on customers that have non-trivial preferences, yet can be fully described by their *types* and *budgets* and do not require any kind of oracles. Assume that a company owns a subset of the links in a given network. The remaining edges are owned by other companies and have fixed publicly known prices and some customer needs to purchase a path between two terminals in the network. Since she is acting rational, she is going to buy the shortest path connecting her terminals. How should we set the prices on the priceable edges in order to maximize the company's revenue? What if there is another customer, who needs to purchase, e.g., a minimum cost spanning tree?

This type of pricing problem, in which preferences are implicitly defined in terms of some optimization problem, is usually referred to as *Stackelberg pricing* [23]. In the standard 2-player form we are given a *leader* setting the prices on a subset of the network and a *follower* seeking to purchase a min-cost network satisfying her requirements. We proceed by formally defining the model before stating our results.

1.1. Model and Notation

In this paper we consider the following class of multi-player one-round games. Let $G = (V, E)$ be a multi-graph. There are two types of players in the game, one *leader* and one or more *followers*. We consider two classes of *edge* and *vertex games*, in which either the edges or the vertices have costs. For most of the paper, we will consider edge games, but the definitions and results for vertex games follow analogously. In an edge game, the edge set E is partitioned into two sets $E = E_p \cup E_f$ with $E_p \cap E_f = \emptyset$. For each *fixed-price* edge $e \in E_f$ there is a fixed cost $c(e) \geq 0$. For each *priceable* edge $e \in E_p$ the leader can specify a price $p(e) \geq 0$. We denote the number of priceable edges by $m = |E_p|$. Each follower $i = 1, \ldots, k$ has a set $\mathcal{S}_i \subset 2^E$ of *feasible subnetworks*. The *weight* $w(S)$ of a subnetwork $S \in \mathcal{S}_i$ is given by the costs of fixed-price edges and the price of priceable edges,

$$w(S) = \sum_{e \in S \cap E_f} c(e) + \sum_{e \in S \cap E_p} p(e).$$

The *revenue* $r(S)$ of the leader from subnetwork S is given by the prices of the priceable edges that are included in S, i.e.,

$$r(S) = \sum_{e \in S \cap E_p} p(e).$$

Throughout the paper we assume that for any price function p every follower i can in polynomial time find a subnetwork $S_i^*(p)$ of minimum weight. Our interest is to find the

pricing function p^* for the leader that generates maximum revenue, i.e.,

$$p^* = \arg\max_p \sum_{i=1}^{k} r(S_i^*(p)).$$

We denote the value of this maximum revenue by r^*. To guarantee that the revenue is bounded and the optimization problem is non-trivial, we assume that there is at least one feasible subnetwork for each follower i that is composed only of fixed-price edges. In order to avoid technicalities, we assume w.l.o.g. that among subnetworks of identical weight the follower always chooses the one with higher revenue for the leader. It is not difficult to see that in the 2-player case we also need followers with a large number of feasible subnetworks in order to make the problem interesting.

Proposition 1.1. *Given follower j and a fixed subnetwork $S_j \in \mathcal{S}_j$, we can compute prices p with $w(S_j) = \min_{S \in \mathcal{S}_j} w(S)$ maximizing $r(S_j)$ or decide that such prices do not exist in polynomial time. In the 2-player game, if $|\mathcal{S}| = \mathcal{O}(poly(m))$, revenue maximization can be done in polynomial time.*

The proof of Proposition 1.1 will appear in the full version. In general we will refer to the revenue optimization problem by Stack. Note that our model extends the previously considered pricing models and is essentially equivalent to pricing with general valuation functions, a problem that has independently been considered in [4]. Every general valuation function can be expressed in terms of Stackelberg network pricing on graphs, and our algorithmic results apply in this setting as well.

1.2. Previous Work and New Results

The single-follower shortest path Stackelberg pricing problem (StackSP) has first been considered by Labbé et al. [17], who derive a bilevel LP formulation of the problem and prove NP-hardness. Roch et al. [19] present a first polynomial time approximation algorithm with a provable performance guarantee, which yields logarithmic approximation ratios. Bouhtou et al. [5] extend the problem to multiple (weighted) followers and present algorithms for a restricted shortest path problem on parallel links. For an overview of most of the initial work on Stackelberg network pricing the reader is referred to [22]. A different line of research has been investigating the application of Stackelberg pricing to network congestion games in order to obtain low congestion Nash equilibria for sets of selfish followers [11, 20, 21].

More recently, Cardinal et al. [9] initiated the investigation of the corresponding minimum spanning tree (StackMST) game, again obtaining a logarithmic approximation guarantee and proving APX-hardness. Their *single-price algorithm*, which assigns the same price to all priceable edges, turns out to be even more widely applicable and yields similar approximation guarantees for any matroid based Stackelberg game.

The first result of our paper is a generalization of this result to general Stackelberg games. The previous limitation to matroids stems from the difficulty to determine the necessarily polynomial number of candidate prices that can be tested by the algorithm. We develop a novel characterization of the small set of *threshold prices* that need to be tested and obtain a polynomial time $(1+\varepsilon)H_m$-approximation (where H_m denotes the m'th harmonic number) for arbitrary $\varepsilon > 0$, which turns out to be perfectly tight for shortest path as well as minimum spanning tree games. This result is found in Section 2.

We then extend the analysis to multiple followers, in which case the approximation ratio becomes $(1+\varepsilon)(H_k + H_m)$. This can be shown to be essentially best possible by an approximation preserving reduction from single-minded combinatorial pricing [13]. Extending the problem even further, we also look at the case of multiple *weighted* followers, which arises naturally in network settings where different followers come with different routing demands. It has been conjectured before that no approximation essentially better than the number of followers is possible in this scenario. We disprove this conjecture by presenting an alternative analysis of the single-price algorithm resulting in an approximation ratio of $(1+\varepsilon)m^2$. Additionally, we derive a lower bound of $\mathcal{O}(m^{\varepsilon})$ for the weighted player case. This resolves a previously open problem from [5]. The results on multiple followers are found in Section 3.

The generic reduction from single-minded to Stackelberg pricing yields a class of networks in which we can price the vertices on one side of a bipartite graph and players aim to purchase minimum cost vertex covers for their sets of edges. This motivates us to return to the classical Stackelberg setting and consider the 2-player bipartite vertex cover game (StackVC). As it turns out, this variation of the game allows polynomial-time algorithms for exact revenue maximization using non-trivial algorithmic techniques. We first present an upper bound on the possible revenue in terms of the min-cost vertex cover not using any priceable vertices and the minimum portion of fixed cost in any possible cover. Using iterated max-flow computations, we then determine a pricing with total revenue that eventually coincides with our upper bound. These results are found in Section 4.

Finally, Section 5 concludes and presents several intriguing open problems for further research. Some of the proofs have been omitted due to space limitations.

2. A Single-Price Algorithm for a Single Follower

Let us assume that we are faced with a single follower and let c_0 denote the cost of a cheapest feasible subnetwork for the follower not containing any of the priceable edges. Clearly, we can compute c_0 by assigning price $+\infty$ to all priceable edges and simulating the follower on the resulting network. The *single-price algorithm* proceeds as follows. For $j = 0, \ldots, \lceil \log c_0 \rceil$ it assigns price $p_j = (1+\varepsilon)^j$ to all priceable edges and determines the resulting revenue $r(p_j)$. It then simply returns the pricing that results in maximum revenue. We present a logarithmic bound on the approximation guarantee of the single-price algorithm.

Theorem 2.1. *Given any $\varepsilon > 0$, the single-price algorithm computes an $(1+\varepsilon)H_m$-approximation with respect to r^*, the revenue of an optimal pricing.*

2.1. Analysis

The single-price algorithm has previously been applied to a number of different combinatorial pricing problems [1, 15]. The main issue in analyzing its performance guarantee for Stackelberg pricing is to determine the right set of candidate prices. We first derive a precise characterization of these candidates and then argue that the geometric sequence of prices tested by the algorithm is a good enough approximation. Slightly abusing notation,

we let p refer to both price p and the assignment of this price to all priceable edges. If there exists a feasible subnetwork for the follower that uses at least j priceable edges, we let

$$\theta_j = \max\Big\{p \;\Big|\; |S^\star(p) \cap E_p| \geq j\Big\}$$

be the largest price at which such a subnetwork is chosen. If no feasible subnetwork with at least j priceable edges exists, we set $\theta_j = 0$. As we shall see, these thresholds are the key to prove Theorem 2.1.

We want to derive an alternative characterization of the values of θ_j. For each $1 \leq j \leq m$ we let c_j refer to the minimum sum of prices of fixed-price edges in any feasible subnetwork containing at most j priceable edges, formally

$$c_j = \min\Big\{\sum_{e \in S \cap E_f} f_e \;\Big|\; S \in \mathcal{S} \,:\, |S \cap E_p| \leq j\Big\},$$

and $\Delta_j = c_0 - c_j$. For ease of notation let $\Delta_0 = 0$. Consider the set of points $(0, \Delta_0)$, $(1, \Delta_1), \ldots, (m, \Delta_m)$ on the plane. By $\mathcal{H}$ we refer to a minimum selection of points spanning the upper convex hull of the point set. It is a straightforward geometric observation that we can define $\mathcal{H}$ as follows:

Fact 1. Point (j, Δ_j) belongs to $\mathcal{H}$ if and only if $\min_{i<j} \frac{\Delta_j - \Delta_i}{j-i} > \max_{j<k} \frac{\Delta_k - \Delta_j}{k-j}$.

We now return to the candidate prices. By definition we have that $\theta_1 \geq \theta_2 \geq \cdots \geq \theta_m$. We say that θ_j is *true threshold value* if $\theta_j > \theta_{j+1}$, i.e., if at price θ_j the subnetwork chosen by the follower contains exactly j priceable edges. Let $i_1 < i_2 < \cdots < i_\ell$ denote the indices, such that θ_{i_k} are true threshold values and for ease of notation define $i_0 = 0$. For an example, see Figure 1.

Lemma 2.2. *θ_j is true threshold value if and only if (j, Δ_j) belongs to $\mathcal{H}$.*

Proof. "⇒" Let θ_j be true threshold value, i.e., at price θ_j the chosen subnetwork contains exactly j priceable edges. We observe that at any price p the cheapest subnetwork containing j priceable edges has cost $c_j + j \cdot p = c_0 - \Delta_j + j \cdot p$. Thus, at price θ_j it must be the case that $\Delta_j - j \cdot \theta_j \geq \Delta_i - i \cdot \theta_j$ for all $i < j$ and $\Delta_j - j \cdot \theta_j > \Delta_k - k \cdot \theta_j$ for all $j < k$. It follows that

$$\min_{i<j} \frac{\Delta_j - \Delta_i}{j - i} \geq \theta_j > \max_{j<k} \frac{\Delta_k - \Delta_j}{k - j},$$

and, thus, we have that (j, Δ_j) belongs to $\mathcal{H}$.

"⇐" Assume now that (j, Δ_j) belongs to $\mathcal{H}$ and let

$$p = \min_{i<j} \frac{\Delta_j - \Delta_i}{j - i}.$$

Consider any $k < j$. It follows that $\Delta_k - k \cdot p = \Delta_j - j \cdot p - (\Delta_j - \Delta_k) + (j - k)p \leq \Delta_j - j \cdot p$, since $p \leq (\Delta_j - \Delta_k)/(j - k)$ and, thus, the network chosen at price p cannot contain less than j priceable edges. Analogously, let $k > j$. Using $p > (\Delta_k - \Delta_j)/(k - j)$ we obtain $\Delta_k - k \cdot p = \Delta_j - j \cdot p + (\Delta_k - \Delta_j) - (k - j)p < \Delta_j - j \cdot p$, and, thus, the subnetwork chosen at price p contains exactly j priceable edges. We conclude that θ_j is a true threshold. ∎

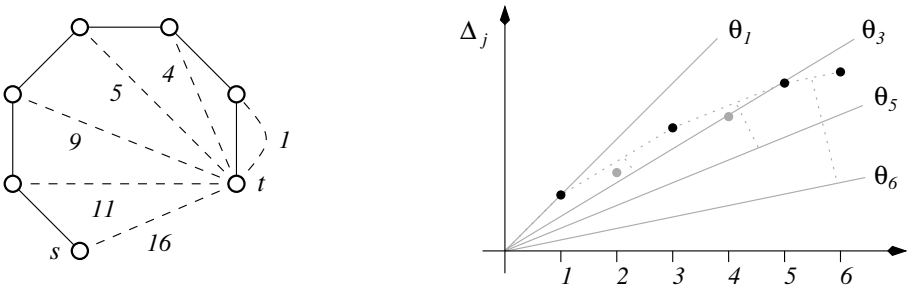


Figure 1: A geometric interpretation of (true) threshold values θ_j. The follower seeks to purchase a shortest path from s to t, dashed edges are fixed-cost.

It is not difficult to see that the price p defined in the second part of the proof of Lemma 2.2 is precisely the threshold value θ_j. Let θ_{i_k} be any true threshold. Since points $(i_0, \Delta_{i_0}), \ldots, (i_\ell, \Delta_{i_\ell})$ define the convex hull we can write that $\min_{i<i_k}(\Delta_{i_k} - \Delta_i)/(i_k - i) = (\Delta_{i_k} - \Delta_{i_{k-1}})/(i_k - i_{k-1})$. We state this important fact again in the following lemma.

Lemma 2.3. *For all $1 \leq k \leq \ell$ it holds that $\theta_{i_k} = \frac{\Delta_{i_k} - \Delta_{i_{k-1}}}{i_k - i_{k-1}}$.*

From the fact that points $(i_0, \Delta_{i_0}), \ldots, (i_\ell, \Delta_{i_\ell})$ define the convex hull we know that $\Delta_{i_\ell} = \Delta_m$, i.e., Δ_{i_ℓ} is the largest of all Δ-values. On the other hand, each Δ_j describes the maximum revenue that can be made from a subnetwork with at most j priceable edges and, thus, Δ_m is clearly an upper bound on the revenue made by an optimal price assignment.

Fact 2. It holds that $r^* \leq \Delta_{i_\ell}$.

By definition of the θ_j's it is clear that at any price below θ_{i_k} the subnetwork chosen by the follower contains no less than i_k priceable edges. Furthermore, for each θ_{i_k} the single-price algorithm tests a candidate price that is at most a factor $(1+\varepsilon)$ smaller than θ_{i_k}. Let $r(p_{i_k})$, $r(\theta_{i_k})$ denote the revenue that results from assigning price p_{i_k} or θ_{i_k} to all priceable edges, respectively.

Fact 3. For each θ_{i_k} there exists a price p_{i_k} with $(1+\varepsilon)^{-1}\theta_{i_k} \leq p_{i_k} \leq \theta_{i_k}$ that is tested by the single-price algorithm. Especially, it holds that $r(p_{i_k}) \geq (1+\varepsilon)^{-1} r(\theta_{i_k})$

Finally, we know that the revenue made by assigning price θ_{i_k} to all priceable edges is $r(\theta_{i_k}) = i_k \cdot \theta_{i_k}$. Let r denote the revenue of the single-price solution returned by the algorithm. We have:

$$\begin{aligned}
(1+\varepsilon)\cdot H_m \cdot r &= (1+\varepsilon)\sum_{j=1}^{m}\frac{r}{j} \geq (1+\varepsilon)\sum_{k=1}^{\ell}\sum_{j=i_{k-1}+1}^{i_k}\frac{r}{j} \geq (1+\varepsilon)\sum_{k=1}^{\ell}\sum_{j=i_{k-1}+1}^{i_k}\frac{r(p_{i_k})}{j} \\
&\geq \sum_{k=1}^{\ell}\sum_{j=i_{k-1}+1}^{i_k}\frac{r(\theta_{i_k})}{j} \geq \sum_{k=1}^{\ell}\sum_{j=i_{k-1}+1}^{i_k}\frac{i_k\cdot\theta_{i_k}}{j} \\
&\geq \sum_{k=1}^{\ell}(i_k - i_{k-1})\frac{i_k\cdot\theta_{i_k}}{i_k} = \sum_{k=1}^{\ell}(\Delta_{i_k} - \Delta_{i_{k-1}}) \text{ , by Lemma 2.3} \\
&= \Delta_{i_\ell} - \Delta_0 = \Delta_{i_\ell} \geq r^*.
\end{aligned}$$

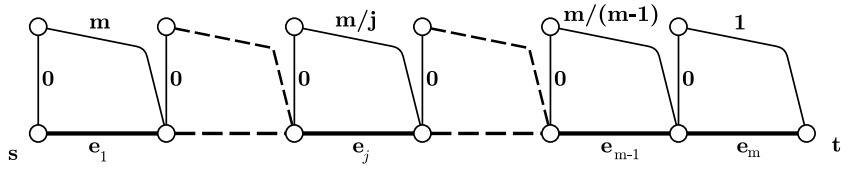


Figure 2: An instance of Stackelberg Shortest Path, on which the analysis of the approximation guarantee of the single-price algorithm is tight. Bold edges are priceable, vertex labels of regular edges indicate cost. The instance yields tightness of the analysis also for Stackelberg Minimum Spanning Tree.

This concludes the proof of Theorem 2.1.

2.2. Tightness

The example in Figure 2 shows that our analysis of the single-price algorithm's approximation guarantee is tight. The follower wants to buy a path connecting vertices s and t. In an optimal solution we set the price of edge e_j to m/j. Then edges $e_1, \ldots, e_m$ form a shortest path of cost mH_m. On the other hand, assume that all edges $e_1, \ldots, e_m$ are assigned the same price p. Every choice will lead to a revenue of at most m. Similar results apply if the follower purchases a minimum spanning tree instead of a shortest path.

The best known lower bound for 2-player Stackelberg pricing is found in [9], where APX-hardness is shown for the minimum spanning tree case. To the authors' best knowledge, up to now no non-constant inapproximability results have been proven. We proceed by extending our results to multiple followers, in which case previous results on other combinatorial pricing problems yield strong lower bounds.

3. Extension to Multiple Followers

In this section we extend our results on general Stackelberg network pricing to scenarios with multiple followers. Recall that each follower j is characterized by her own collection $\mathcal{S}_j$ of feasible subnetworks and k denotes the number of followers. Section 3.1 extends the analysis from the single follower case to prove a tight bound of $(1+\varepsilon)(H_k + H_m)$ on the approximation guarantee of the single-price algorithm. In addition, it presents an alternative analysis that applies even in the case of weighted followers and yields approximation guarantees that do not depend on the number of followers. Section 3.2 derives (near) tight inapproximability results based on known hardness results for combinatorial pricing. Proofs are omitted due to space limitations.

3.1. Guarantees of the Single-Price Algorithm

Let an instance of Stackelberg network pricing with some number $k \geq 1$ of followers be given. We obtain a similar bound on the single-price algorithm's approximation guarantee.

Theorem 3.1. *The single-price algorithm computes an $(1+\varepsilon)(H_k + H_m)$-approximation with respect to r^*, the revenue of an optimal pricing, for* Stack *with multiple followers.*

The proof of Theorem 3.1 reduces the problem to the single player case. However, it relies essentially on the fact that we are considering the single-price algorithm. It does not imply anything about the relation of these two cases in general.

An even more general variation of Stackelberg pricing, in which we allow multiple *weighted* followers, arises naturally in the context of network pricing games with different demands for each player. This model has been previously considered in [5]. Formally, for each follower j we are given her *demand* $d_j \in \mathbb{R}_0^+$. Given followers buying subnetworks $S_1, \ldots, S_k$, the leader's revenue is defined as $\sum_{j=1}^{k} d_j \sum_{e \in S_j \cap E_p} p(e)$. It has been conjectured before that in the weighted case no approximation guarantee essentially beyond $\mathcal{O}(k \cdot \log m)$ is possible [19]. We show that an alternative analysis of the single-price algorithm yields ratios that do not depend on the number of followers.

Theorem 3.2. *The single-price algorithm computes an $(1+\varepsilon)m^2$-approximation with respect to r^*, the revenue of an optimal pricing, for* Stack *with multiple weighted followers.*

3.2. Lower Bounds

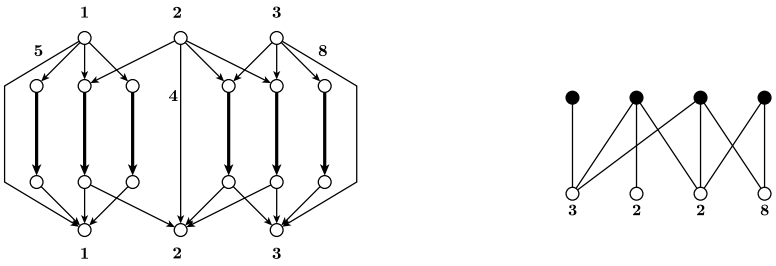
Hardness of approximation of Stackelberg pricing with multiple followers follows immediately from known results about other combinatorial pricing models. Theorem 3.3 is based on a reduction from the (weighted) unit-demand envy-free pricing problem with uniform budgets, which is known to be inapproximable within $\mathcal{O}(m^{\varepsilon})$ (m denotes the number of products) [6]. Here we are given a universe of products and a collection of (weighted) customers, each of which buys the cheapest product out of some set of alternatives with a price not exceeding her budget. The resulting Stackelberg game is an instance of the so-called *river tariffication problem*. Each player needs to route her demand along one out of a number of parallel links connecting her respective source and sink pair. One direct fixed price connection determines her maximum budget for purchasing a priceable link. Theorem 3.3 resolves an open problem from [5]. The construction is depicted in Figure 3(a).

Theorem 3.3. *The Stackelberg network pricing problem with multiple weighted followers is hard to approximate within $\mathcal{O}(m^{\varepsilon})$ for some $\varepsilon > 0$, unless NP $\subseteq \bigcap_{\delta>0}$ BPTIME($2^{n^{\delta}}$). The same holds for the river tariffication problem.*

Theorem 3.4 is based on a reduction from the single-minded combinatorial pricing problem, in which each customer is interested in a subset of products and purchases the whole set if the sum of prices does not exceed her budget. Single-minded pricing is hard to approximate within $\mathcal{O}(\log^{\varepsilon} k + \log^{\varepsilon} m)$ [13], where k and m denote the numbers of customers and products, respectively. Theorem 3.4 shows that the single-price algorithm is essentially best possible for multiple unweighted followers.

Theorem 3.4. *The Stackelberg network pricing problem with multiple unweighted followers is hard to approximate within $\mathcal{O}(\log^{\varepsilon} k + \log^{\varepsilon} m)$ for some $\varepsilon > 0$, unless NP $\subseteq \bigcap_{\delta>0}$ BPTIME($2^{n^{\delta}}$). The same holds for bipartite Stackelberg Vertex Cover Pricing (*StackVC*).*

The idea for the proof of Theorem 3.4 is illustrated in Figure 3(b). We define an instance of StackVC in bipartite graphs. Vertices on one side of the bipartition are priceable and represent the universe of products, vertices on the other side encode customers and have fixed prices corresponding to the respective budgets. For each customer we define a follower in the Stackelberg game with edges connecting the customer vertex and all product vertices



(a) (b)

Figure 3: Reductions from pricing problems to Stackelberg pricing. (a) Unit-demand reduces to directed StackSP. Bold edges are priceable, edge labels indicate cost. Regular edges without labels have cost 0. Vertex labels indicate source-sink pairs for the followers. (b) Single-minded pricing reduces to bipartite StackVC. Filled vertices are priceable, vertex labels indicate cost. For each customer there is one follower, who strives to cover all incident edges.

the customer wishes to purchase. Now every follower seeks to buy a min-cost vertex cover for her set of edges. We proceed by taking a closer look at this special type of Stackelberg pricing game and especially focus on the interesting case of a single follower.

4. Stackelberg Vertex Cover

Stackelberg Vertex Cover Pricing is a vertex game, however, the approximation results for the single-price algorithm continue to hold. Note that in general the vertex cover problem is hard, hence we focus on settings, in which the problem can be solved in polynomial time. In bipartite graphs the problem can be solved optimally by using a classic and fundamental max-flow/min-cut argumentation. If all priceable vertices are in one side of the partition, then for multiple followers there is evidence that the single-price algorithm is essentially best possible. Our main theorem in this section states that the setting with a single follower can be solved exactly. As a consequence, general bipartite StackVC can be approximated by a factor of 2.

Theorem 4.1. *If for a bipartite graph $G = (A \cup B, E)$ we have $V_p \subseteq A$, then there is a polynomial time algorithm computing an optimal price function p^* for* StackVC.

Before we prove the theorem, we mention that the standard problem of minimum vertex cover in a bipartite graph G with disjoint vertex sets A, B and edges $E \subseteq A \times B$ can be solved by the following application of LP-duality. The LP-dual is interpreted as a maximum flow problem on an adjusted flow network G_d. In particular, G_d is constructed by adding a source s and a sink t to G and connecting s to all vertices $v \in A$ with directed edges (s, v), and t to all vertices $v \in B$ with directed edges (v, t). Each such edge gets as capacity the cost of the involved original vertex - i.e. $p(v)$ for $v \in V_p$ or $c(v)$ if $v \in V_f$. Furthermore, all original edges of the graph are directed from A to B and their capacity is set to infinity. The value of a maximum s-t-flow equals the cost of a minimum cut, and in addition the cost of a minimum cost vertex cover of the graph G (for an example see Figure 4). To obtain such a cover consider an *augmenting s-t-path* in G_d, which is a path traversing only forward edges with slack capacity and backward edges with non-zero flow. The maximum

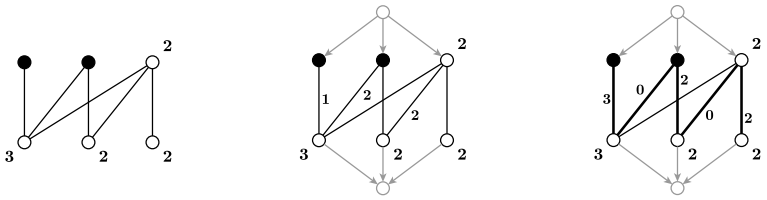


Figure 4: Construction to solve bipartite StackVC with priceable vertices in one partition and a single follower. Filled vertices are priceable, vertex labels indicate cost. (a) A graph G; (b) The flow network G_d obtained from G. Grey parts are source and sink added by the transformation. Edge labels indicate a suboptimal s-t-flow; (c) An augmenting path P indicated by bold edges and the resulting flow. Every such path P starts with a priceable vertex, and all priceable vertices remain in the optimum cover at all times.

flow can be computed by iteratively increasing flow along such paths. The vertices in the minimum vertex cover then correspond to incident edges in a minimum cut. In particular, the minimum vertex cover includes a vertex $v \in A$ if the flow allows no augmenting s-v-path from s to v, i.e. if every path from s to v has at least one backward edge with no flow, or at least one forward edge without slack capacity.

We use a similar idea to obtain the optimal pricing for StackVC. Let $n = |V_p|$ and the values c_j for $1 \leq j \leq n$ denote the minimum sum of prices of fixed-price vertices in any feasible subnetwork containing at most j priceable vertices. Then, $\Delta_j = c_0 - c_j$ are again upper bounds on the revenue that can be extracted from a network that includes at most j priceable vertices. We thus have $r^* \leq \Delta_n$.

Algorithm 1: Solving StackVC in bipartite graphs with $V_p \subseteq A$

```
1 Construct the flow network G_d by adding nodes s and t
2 Set p(v) = 0 for all v ∈ V_p
3 Compute a maximum s-t-flow φ in G_d
4 while there is v ∈ V_p s.t. increasing p(v) yields an augmenting s-t-path P do
5     Increase p(v) and φ along P as much as possible
```

Suppose all priceable vertices are located in one partition $V_p \subseteq A$ and consider Algorithm 1. We denote by $\mathcal{C}_{ALG}$ the cover calculated by Algorithm 1. At first, when computing the maximum flow on G_d holding all $p(v) = 0$, the algorithm obtains a flow of c_n. We first note that in the following while-loop we will never face a situation, in which there is an augmenting s-t-path (traversing forward edges with slack capacity and backward edges with non-zero flow) starting with a fixed-price vertex. We call such a path a *fixed* path, while an augmenting s-t-path starting with a priceable vertex is called a *price* path.

Lemma 4.2. *Every augmenting path considered in the while-loop of Algorithm 1 is a price path.*

Proof. We prove the lemma by induction on the while-loop and by contradiction. Suppose that in the beginning of the current iteration there is no fixed path. In particular, this is

true for the first iteration of the while-loop. Then, suppose that after we have increased the flow over a price path P_p, a fixed path P_f is created. P_f must include some of the edges of P_p. Consider the vertex w at which P_f hits P_p. By following P_f from s to w and P_p from w to t there is a fixed path, which must have been present before flow was increased on P_p. This is a contradiction and proves the lemma. ∎

Recall from above that the optimum cover contains a vertex $v \in A$ if there is no augmenting s-v-path from s to v. In particular, this means that for a vertex $v \in A \cap \mathcal{C}$ the following two properties are fulfilled: (1) there is no slack capacity on edge (s, v); (2) there is no augmenting s-v-path from s over a different vertex $v' \in A$. As the algorithm always adjusts the price of a vertex v to equal the current flow on (s, v), only the violation of property (2) can force a vertex $v \in V_p$ to leave the cover. In particular, such an augmenting s-v-path must start with a fixed-price vertex, and it must reach v by decreasing flow over one of the original edges (v, w) for $w \in B$. We call such a path a *fixed v-path*.

Lemma 4.3. *Algorithm 1 creates no fixed v-path for any priceable vertex $v \in V_p$.*

The proof of Lemma 4.3 is similar to the proof of Lemma 4.2 and will appear in the full version. As there is no augmenting path from s to any priceable vertex at any time, the following lemma is now obvious.

Lemma 4.4. *$\mathcal{C}_{ALG}$ includes all priceable vertices.*

Proof of Theorem 4.1. Finally, we can proceed to argue that the computed pricing is optimal. Suppose that after executing Algorithm 1 we increase $p(v)$ over $\phi(s, v)$ for any priceable vertex v. As we are at the end of the algorithm, it does not allow us to increase the flow in the same way. Thus, the adjustment creates slack capacity on all the edges (s, v) for any $v \in V_p$ and causes every priceable vertex to leave $\mathcal{C}_{ALG}$. The new cover must be the cheapest cover that excludes every priceable vertex, i.e. it must be $\mathcal{C}_0$ and have cost c_0. As we have not increased the flow, we know that the cost of $\mathcal{C}_{ALG}$ is also c_0. Note that before starting the while-loop the cover was $\mathcal{C}_n$ of cost c_n. As all flow increase in the while-loop was made over price paths and all the priceable vertices stay in the cover, the revenue of $\mathcal{C}_{ALG}$ must be $c_0 - c_n = \Delta_n$. This is an upper bound on the optimum revenue, and hence the price function p_{ALG} derived with the algorithm is optimal. Finally, notice that adjusting the price of the priceable vertices in each iteration is not necessary. We can start with computing $\mathcal{C}_n$ and for the remaining while-loop set all prices to $+\infty$. This will result in the desired flow, which directly generates the final price for every vertex v as flow on (s, v). Hence, we can get optimal prices with an adjusted run of the standard polynomial time algorithm for maximum flow in G_d. This proves Theorem 4.1. ∎

Theorem 4.5. *There is a polynomial time 2-approximation algorithm for bipartite* StackVC.

In Theorem 4.5 we use the previous analysis to get a 2-approximation of the optimum revenue for general bipartite StackVC. This results in a $2k$-approximation for any number of k followers. In contrast, the analysis of the single-price algorithm is tight even for one follower and all priceable vertices in one partition. Moreover, bipartite StackVC for at least two followers is NP-hard by a reduction from the highway pricing problem [7].

5. Open problems

There are a number of important open problems that arise from our work. We believe that the single-price algorithm is essentially best possible even for a single follower and

general Stackelberg pricing games. However, there is no matching logarithmic lower bound, and the best lower bound remains APX-hardness from [9]. In addition, we believe that for weighted followers a better upper bound than m^2 is possible, which would decrease the gap to the $\Omega(m^{\varepsilon})$ lower bound we observed. More generally, extending other algorithm design techniques to cope with pricing problems is a major open problem.

References

[1] G. Aggarwal, T. Feder, R. Motwani, and A. Zhu. Algorithms for Multi-Product Pricing. In *Proc. of 31st ICALP*, 2004.

[2] N. Balcan and A. Blum. Approximation Algorithms and Online Mechanisms for Item Pricing. In *Proc. of 7th EC*, 2006.

[3] N. Balcan, A. Blum, J. Hartline, and Y. Mansour. Mechanism Design via Machine Learning. In *Proc. of 46th FOCS*, 2005.

[4] M. Balcan, A. Blum, and Y. Mansour. Single Price Mechanisms for Revenue Maximization in Unlimited Supply Combinatorial Auctions. Technical Report *CMU-CS-07-111*, Carnegie Mellon University, 2007.

[5] M. Bouhtou, A. Grigoriev, S. van Hoesel, A. van der Kraaij, F. Spieksma, and M. Uetz. Pricing Bridges to Cross a River. *Naval Research Logistics*, 54(4): 411–420, 2007.

[6] P. Briest. Towards Hardness of Envy-Free Pricing. *ECCC Technical Report TR06-150*, 2006.

[7] P. Briest and P. Krysta. Single-Minded Unlimited-Supply Pricing on Sparse Instances. In *Proc. of 17th SODA*, 2006.

[8] P. Briest and P. Krysta. Buying Cheap is Expensive: Hardness of Non-Parametric Multi-Product Pricing. In *Proc. of 18th SODA*, 2007.

[9] J. Cardinal, E. Demaine, S. Fiorini, G. Joret, S. Langerman, I. Newman, and O. Weimann. The Stackelberg Minimum Spanning Tree Game. In *Proc. of 10th WADS*, 2007.

[10] S. Chawla, J. Hartline, and R. Kleinberg. Algorithmic Pricing via Virtual Valuations. In *Proc. of 8th EC*, 2007.

[11] R. Cole, Y. Dodis, and T. Roughgarden. Pricing Network Edges for Heterogeneous Selfish Users. In *Proc. of 35th STOC*, 2003.

[12] P. Cramton, Y. Shoham, and R. Steinberg (Editors). *Combinatorial Auctions.* MIT Press, 2006.

[13] E.D. Demaine, U. Feige, M.T. Hajiaghayi, and M.R. Salavatipour. Combination Can Be Hard: Approximability of the Unique Coverage Problem. In *Proc. of 17th SODA*, 2006.

[14] P. Glynn, P. Rusmevichientong, and B. Van Roy. A Non-Parametric Approach to Multi-Product Pricing. *Operations Research*, **54**(1):82–98, 2006.

[15] V. Guruswami, J.D. Hartline, A.R. Karlin, D. Kempe, C. Kenyon, and F. McSherry. On Profit-Maximizing Envy-Free Pricing. In *Proc. of 16th SODA*, 2005.

[16] J. Hartline and V. Koltun. Near-Optimal Pricing in Near-Linear Time. In *Proc. of 8th WADS*, 2005.

[17] M. Labbé, P. Marcotte, and G. Savard. A Bilevel Model of Taxation and its Application to Optimal Highway Pricing. *Management Science*, 44(12): 1608–1622, 1998.

[18] N. Nisan and A. Ronen. Algorithmic Mechanism Design. In *Proc. of 31st STOC*, 1999.

[19] S. Roch, G. Savard, and P. Marcotte. An Approximation Algorithm for Stackelberg Network Pricing. *Networks*, 46(1): 57–67, 2005.

[20] T. Roughgarden. Stackelberg Scheduling Strategies. *SIAM J. on Computing*, 33(2): 332–350, 2004.

[21] C. Swamy. The Effectiveness of Stackelberg Strategies and Tolls for Network Congestion Games. In *Proc. of 18th SODA*, 2007.

[22] S. van Hoesel. An Overview of Stackelberg Pricing in Networks. Research Memoranda 042, METEOR, Maastricht, 2006.

[23] H. von Stackelberg. *Marktform und Gleichgewicht (Market and Equilibrium).* Verlag von Julius Springer, Vienna, 1934.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 145-156
www.stacs-conf.org

SUBLINEAR COMMUNICATION PROTOCOLS FOR MULTI-PARTY POINTER JUMPING AND A RELATED LOWER BOUND

JOSHUA BRODY [1] AND AMIT CHAKRABARTI [1]

[1] Department of Computer Science
Dartmouth College
Hanover, NH 03755, USA

Abstract. We study the one-way number-on-the-forehead (NOF) communication complexity of the k-layer pointer jumping problem with n vertices per layer. This classic problem, which has connections to many aspects of complexity theory, has seen a recent burst of research activity, seemingly preparing the ground for an $\Omega(n)$ lower bound, for constant k. Our first result is a surprising sublinear — i.e., $o(n)$ — upper bound for the problem that holds for $k \geq 3$, dashing hopes for such a lower bound.

A closer look at the protocol achieving the upper bound shows that all but one of the players involved are *collapsing*, i.e., their messages depend only on the composition of the layers ahead of them. We consider protocols for the pointer jumping problem where *all* players are collapsing. Our second result shows that a strong $n - O(\log n)$ lower bound does hold in this case. Our third result is another upper bound showing that nontrivial protocols for (a non-Boolean version of) pointer jumping are possible even when all players are collapsing.

Our lower bound result uses a novel proof technique, different from those of earlier lower bounds that had an information-theoretic flavor. We hope this is useful in further study of the problem.

1. Introduction

Multi-party communication complexity in general, and the *pointer jumping* problem (also known as the *pointer chasing* problem) in particular, has been the subject of plenty of recent research. This is because the model, and sometimes the specific problem, bears on several aspects of computational complexity: among them, circuit complexity [Yao90, HG91, BT94], proof size lower bounds [BPS05] and space lower bounds for streaming algorithms [AMS99, GM07, CJP08]. The most impressive known consequence of a strong

1998 ACM Subject Classification: F.1.3, F.2.2.

Key words and phrases: Communication complexity, pointer jumping, number on the forehead.

Work supported in part by an NSF CAREER Award CCF-0448277, NSF grants CCF-0514870 and EIA-98-02068. Work partly done while the authors were visiting the University of Washington, Seattle, WA.
.

multi-party communication lower bound would be to exhibit non-membership in the complexity class ACC^0; details can be found in Beigel and Tarui [BT94] or in the textbook by Arora and Barak [AB07]. Vexingly, it is not even known whether or not $\mathsf{ACC}^0 = \mathsf{NEXP}$.

The setting of multi-party communication is as follows. There are k players (for some $k \geq 2$), whom we shall call $\text{PLR}_1, \text{PLR}_2, \ldots, \text{PLR}_k$, who share an input k-tuple $(x_1, x_2, \ldots, x_k)$. The goal of the players is to compute some function $f(x_1, x_2, \ldots, x_k)$. There are two well-studied sharing models: the *number-in-hand* model, where PLR_i sees x_i, and the *number-on-the-forehead* (NOF) model, where PLR_i sees all x_js such that $j \neq i$. Our focus in this paper will be on the latter model, which was first introduced by Chandra, Furst and Lipton [CFL83]. It is in this model that communication lower bounds imply lower bounds against ACC^0. We shall use $C(f)$ to denote the deterministic communication complexity of f in this model. Also of interest are randomized protocols that only compute $f(x)$ correctly with high probability: we let $R_\varepsilon(f)$ denote the ε-error randomized communication complexity of f. Our work here will stick to deterministic protocols, which is a strength for our upper bounds. Moreover, it is not a serious weakness for our lower bound, because the ACC^0 connection only calls for a deterministic lower bound.

Notice that the NOF model has a feature not seen elsewhere in communication complexity: the players *share* plenty of information. In fact, for large k, each individual player already has "almost" all of the input. This intuitively makes lower bounds especially hard to prove and indeed, to this day, no nontrivial lower bound is known in the NOF model for any explicit function with $k = \omega(\log n)$ players, where n is the total input size. The pointer jumping problem is widely considered to be a good candidate for such a lower bound. As noted by Damm, Jukna and Sgall [DJS98], it has many natural special cases, such as shifting, addressing, multiplication and convolution. This motivates our study.

1.1. The Pointer Jumping Problem and Previous Results

There are a number of variants of the pointer jumping problem. Here we study two variants: a Boolean problem, MPJ_k^n, and a non-Boolean problem, $\widehat{\text{MPJ}}_k^n$ (henceforth, we shall drop the superscript n). In both variants, the input is a subgraph of a fixed layered graph that has $k+1$ layers of vertices, with layer 0 consisting of a single vertex, v_0, and layers 1 through $k-1$ consisting of n vertices each (we assume $k \geq 2$). Layer k consists of 2 vertices in the case of MPJ_k and n vertices in the case of $\widehat{\text{MPJ}}_k$. The input graph is a subgraph of the fixed layered graph in which every vertex (except those in layer k) has outdegree 1. The desired output is the name of the unique vertex in layer k reachable from v_0, i.e., the final result of "following the pointers" starting at v_0. The output is therefore a single bit in the case of MPJ_k or a $\lceil \log n \rceil$-bit string in the case of $\widehat{\text{MPJ}}_k$.[1]

The functions MPJ_k and $\widehat{\text{MPJ}}_k$ are made into NOF communication problems as follows: for each $i \in [k]$, a description of the ith layer of edges (i.e., the edges pointing into the ith layer of vertices) is written on PLR_i's forehead. In other words, PLR_i sees every layer of edges except the ith. The players are allowed to write one message each on a public *blackboard* and must do so in the fixed order $\text{PLR}_1, \text{PLR}_2, \ldots, \text{PLR}_k$. The final player's message must be the desired output. Notice that the specific order of speaking — $\text{PLR}_1, \text{PLR}_2, \ldots, \text{PLR}_k$ — is important to make the problem nontrivial. Any other order of speaking allows an easy deterministic protocol with only $O(\log n)$ communication.

[1] Throughout this paper we use "log" to denote logarithm to the base 2.

Consider the case $k = 2$. The problem MPJ_2 is equivalent to the two-party communication problem INDEX, where Alice holds a bit-vector $x \in \{0,1\}^n$, Bob holds an index $i \in [n]$, and Alice must send Bob a message that enables him to output x_i. It is easy to show that $C(\text{MPJ}_2) = n$. In fact, Ablayev [Abl96] shows the tight tradeoff $R_\varepsilon(\text{MPJ}_2) = (1 - H(\varepsilon))n$, where H is the binary entropy function. It is tempting to conjecture that this lower bound generalizes as follows.

Conjecture 1.1. There is a nondecreasing function $\xi : \mathbb{Z}^+ \to \mathbb{R}^+$ such that, $\forall\, k :\ C(\text{MPJ}_k) = \Omega(n/\xi(k))$.

Note that, by the results of Beigel and Tarui [BT94], in order to show that $\text{MPJ}_k \notin \mathsf{ACC}^0$ it would suffice, for instance, to prove the following (possibly weaker) conjecture.

Conjecture 1.2. There exist constants $\alpha, \beta > 0$ such that, for $k = n^\alpha$, $C(\text{MPJ}_k) = \Omega(n^\beta)$.

Conjecture 1.1 is consistent with (and to an extent motivated by) research prior to this work. In weaker models of information sharing than the NOF model, an equivalent statement is known to be true, even for randomized protocols. For instance, Damm, Jukna and Sgall [DJS98] show an $\Omega(n/k^2)$ communication lower bound in the so-called *conservative* model, where PLR_i has only a limited view of the layers of the graph behind her: she only sees the result of following the first $i-1$ pointers. Chakrabarti [Cha07] extends this bound to randomized protocols and also shows an $\Omega(n/k)$ lower bound in the so-called *myopic* model, where PLR_i has only a limited view of the layers ahead of her: she cannot see layers $i+2, \ldots, k$.

For the full NOF model, Wigderson, building on the work of Nisan and Wigderson [NW93], showed that $C(\text{MPJ}_3) = \Omega(\sqrt{n})$. This result is unpublished, but an exposition can be found in Babai, Hayes and Kimmel [BHK01]. Very recently, Viola and Wigderson [VW07] generalized this result and extended it to randomized protocols, showing that $R_{1/3}(\text{MPJ}_k) = \Omega(n^{1/(k-1)}/k^{O(k)})$. Of course, this bound falls far short of that in Conjecture 1.1 and does nothing for Conjecture 1.2. However, it is worth noting that the Viola-Wigderson bound in fact applies to the much smaller subproblem of *tree pointer jumping* (denoted TPJ_k), where the underlying layered graph is a height-k tree, with every vertex in layers 0 through $k-2$ having $n^{1/(k-1)}$ children and every vertex in layer $k-1$ having two children. It is easy to see that $C(\text{TPJ}_k) = O(n^{1/(k-1)})$. Thus, one might hope that the more general problem MPJ_k has a much stronger lower bound, as in Conjecture 1.1.

On the upper bound side, Damm et al. [DJS98] show that $C(\widehat{\text{MPJ}}_k) = O(n \log^{(k-1)} n)$, where $\log^{(i)} n$ is the ith iterated logarithm of n. This improves on the trivial upper bound of $O(n \log n)$. Their technique does not yield anything nontrivial for the Boolean problem MPJ_k, though. However, Pudlak, Rödl and Sgall [PRS97] obtain a sublinear upper bound of $O(n \log\log n / \log n)$ for a special case of MPJ_3. Their protocol works only when every vertex in layer 2 has *indegree* 1, or equivalently, when the middle layer of edges in the input describes a *permutation* of $[n]$.

1.2. Our Results

The protocol of Pudlak et al. [PRS97] did not rule out Conjecture 1.1, but it did suggest caution. Our first result is the following upper bound — in fact the first nontrivial upper bound on $C(\text{MPJ}_k)$ — that falsifies the conjecture.

Theorem 1.3. *For $k \geq 3$, we have*

$$C(\text{MPJ}_k) \;=\; O\left(n\left(\frac{k\log\log n}{\log n}\right)^{(k-2)/(k-1)}\right).$$

In particular, $C(\text{MPJ}_3) = O(n\sqrt{\log\log n/\log n})$.

A closer look at the protocol that achieves the upper bound above reveals that all players except for PLR_1 behave in the following way: the message sent by PLR_i depends only on layers 1 through $i-1$ and the composition of layers $i+1$ through k. We say that PLR_i is *collapsing*. This notion is akin to that of the aforementioned conservative protocols considered by Damm et al. Whereas a conservative player composes the layers behind hers, a collapsing player does so for layers ahead of hers.

We consider what happens if we require *all* players in the protocol to be collapsing. We prove a strong linear lower bound, showing that even a single non-collapsing player makes an asymptotic difference in the communication complexity.

Theorem 1.4. *In a protocol for MPJ_k where every player is collapsing, some player must communicate at least $n - \frac{1}{2}\log n - 2 = n - O(\log n)$ bits.*

Finally, one might wonder whether the collapsing requirement is so strong that nothing nontrivial is possible anyway. The same question can be raised for the conservative and myopic models where $\Omega(n/k^2)$ and $\Omega(n/k)$ lower bounds were proven in past work. It turns out that the upper bound on $C(\widehat{\text{MPJ}}_k)$ due to Damm et al. [DJS98] (see Section 1.1) is achievable by a protocol that is both conservative and myopic. We can show a similar upper bound via a different protocol where every player is collapsing.

Theorem 1.5. *For $k \geq 3$, there is an $O(n\log^{(k-1)} n)$-communication protocol for $\widehat{\text{MPJ}}_k^{\text{perm}}$ in which every player is collapsing. Here $\widehat{\text{MPJ}}_k^{\text{perm}}$ denotes the subproblem of $\widehat{\text{MPJ}}_k$ in which layers 2 through k of the input graph are permutations of $[n]$.*

The requirement that layers be permutations is a natural one and is not new. The protocol of Pudlak et al. also had this requirement; i.e., it gave an upper bound on $C(\text{MPJ}_3^{\text{perm}})$. Theorem 1.5 can in fact be strengthened slightly by allowing one of the layers from 2 through k to be arbitrary; we formulate and prove this stronger version in Section 4.

1.3. Organization

The rest of the paper is organized as follows. Theorems 1.3, 1.4 and 1.5 are proven in Sections 2, 3 and 4 respectively. Section 2.1 introduces some notation that is used in subsequent sections.

2. A Sublinear Upper Bound

2.1. Preliminaries, Notation and Overall Plan

For the rest of the paper, "protocols" will be assumed to be deterministic one-way NOF protocols unless otherwise qualified. We shall use $\text{cost}(P)$ to denote the total number of bits communicated in P, for a worst case input.

Let us formally define the problems MPJ_k and $\widehat{\text{MPJ}}_k$. We shall typically write the input k-tuple for MPJ_k as $(i, f_2, \ldots, f_{k-1}, x)$ and that for $\widehat{\text{MPJ}}_k$ as $(i, f_2, \ldots, f_k)$, where $i \in [n]$, each $f_j \in [n]^{[n]}$ and $x \in \{0,1\}^n$. We then define $\text{MPJ}_k : [n] \times \left([n]^{[n]}\right)^{k-2} \times \{0,1\}^n \to \{0,1\}$ and $\widehat{\text{MPJ}}_k : [n] \times \left([n]^{[n]}\right)^{k-1} \to [n]$ as follows.

$$\text{MPJ}_2(i,x) := x_i\,; \qquad \text{MPJ}_k(i, f_2, f_3, \ldots, f_{k-1}, x) := \text{MPJ}_{k-1}(f_2(i), f_3, \ldots, f_{k-1}, x)\,, \text{ for } k \geq 3$$

$$\widehat{\text{MPJ}}_2(i,f) := f(i)\,; \qquad \widehat{\text{MPJ}}_k(i, f_2, f_3, \ldots, f_k) := \widehat{\text{MPJ}}_{k-1}(f_2(i), f_3, \ldots, f_k)\,, \text{ for } k \geq 3\,.$$

Here, x_i denotes the ith bit of the string x. It will be helpful, at times, to view strings in $\{0,1\}^n$ as functions from $[n]$ to $\{0,1\}$ and use functional notation accordingly. It is often useful to discuss the composition of certain subsets of the inputs. Let $\hat{i}_2 := i$, and for $3 \leq j \leq k$, let $\hat{i}_j := f_{j-1} \circ \cdots \circ f_2(i)$. Similarly, let $\hat{x}_{k-1} := x$, and for $1 \leq j \leq k-2$, let $\hat{x}_j := x \circ f_{k-1} \circ \cdots \circ f_{j+1}$. Unrolling the recursion in the definitions, we see that, for $k \geq 2$,

$$\text{MPJ}_k(i, f_2, \ldots, f_{k-1}, x) \;=\; x \circ f_{k-1} \circ \cdots \circ f_2(i) \;=\; \hat{x}_1(i) \;=\; x_{\hat{i}_k}\,; \tag{2.1}$$

$$\widehat{\text{MPJ}}_k(i, f_2, \ldots, f_k) \;=\; f_k \circ \cdots \circ f_2(i) \;=\; f_k(\hat{i}_k)\,. \tag{2.2}$$

We also consider the subproblems $\text{MPJ}_k^{\text{perm}}$ and $\widehat{\text{MPJ}}_k^{\text{perm}}$ where each f_j above is a bijection from $[n]$ to $[n]$ (equivalently, a permutation of $[n]$). We let $\mathcal{S}_n$ denote the set of all permutations of $[n]$.

Here is a rough plan of the proof of our sublinear upper bound. We leverage the fact that a protocol P for $\text{MPJ}_3^{\text{perm}}$ with sublinear communication is known. To be precise:

Fact 2.1 (Pudlak, Rödl and Sgall [PRS97, Corollary 4.8])**.** $C(\text{MPJ}_3^{\text{perm}}) = O(n \log\log n / \log n)$.

The exact structure of P will not matter; we shall only use P as a black box. To get a sense for why P might be useful for, say, MPJ_3, note that the players could replace f_2 with a permutation π and just simulate P, and this would work if $\pi(i) = f(i)$. Of course, there is no way for PLR_1 and PLR_3 to agree on a suitable π without communication. However, as we shall see below, it is possible for them to agree on a small enough *set* of permutations such that either some permutation in the set is suitable, or else only a small amount of side information conveys the desired output bit to PLR_3.

This idea eventually gives us a sublinear protocol for MPJ_3. Clearly, whatever upper bound we obtain for MPJ_3 applies to MPJ_k for all $k \geq 3$. However, we can decrease the upper bound as k increases, by embedding several instances of MPJ_3 into MPJ_k. For clarity, we first give a complete proof of Theorem 1.3 for the case $k = 3$.

2.2. A 3-Player Protocol

Following the plan outlined above, we prove Theorem 1.3 for the case $k = 3$ by plugging Fact 2.1 into the following lemma, whose proof is the topic of this section.

Lemma 2.2. *Suppose $\phi : \mathbb{Z}^+ \to (0,1]$ is a function such that $C(\text{MPJ}_3^{\text{perm}}) = O(n\phi(n))$. Then $C(\text{MPJ}_3) = O(n\sqrt{\phi(n)})$.*

Definition 2.3. A set $\mathcal{A} \subseteq \mathcal{S}_n$ of permutations is said to d-cover a function $f : [n] \to [n]$ if, for each $r \in [n]$, at least one of the following conditions holds:

(i) $\exists\, \pi \in \mathcal{A}$ such that $\pi(r) = f(r)$, or
(ii) $|f^{-1}(f(r))| > d$.

Lemma 2.4. *Let $f : [n] \to [n]$ be a function and d be a positive integer. There exists a set $\mathcal{A}_d(f) \subseteq \mathcal{S}_n$, with $|\mathcal{A}_d(f)| \leq d$, that d-covers f.*

Proof. We give an explicit algorithm to construct $\mathcal{A}_d(f)$. Our strategy is to partition the domain and codomain of f (both of which equal $[n]$) into parts of matching sizes and then define bijections between the corresponding parts. To be precise, suppose $\text{Range}(f) = \{s_1, s_2, \ldots, s_t\}$. Let $A_i = f^{-1}(s_i)$ be the corresponding fibers of f. Clearly, $\{A_i\}_{i=1}^t$ is a partition of $[n]$. It is also clear that there exists a partition $\{B_i\}_{i=1}^t$ of $[n]$ such that, for all $i \in [t]$, $B_i \cap \text{Range}(f) = \{s_i\}$ and $|B_i| = |A_i|$. We shall now define certain bijections $\pi_{i,\ell} : A_i \to B_i$, for each $i \in [t]$ and $\ell \in [d]$.

Let $a_{i,1} < a_{i,2} < \cdots < a_{i,|A_i|}$ be the elements of A_i arranged in ascending order. Similarly, let $b_{i,1} < \cdots < b_{i,|B_i|}$ be those of B_i. We define

$$\pi_{i,\ell}(a_{i,j}) \;:=\; b_{i,(j-\ell) \bmod |B_i|}\,, \quad \text{for } i \in [t], \ell \in [d]\,,$$

where, for convenience, we require "α mod β" to return values in $[\beta]$, rather than $\{0, 1, \ldots, \beta - 1\}$. It is routine to verify that $\pi_{i,\ell}$ is a bijection. Notice that this construction ensures that for all $i \in [t]$ and $j \in [|A_i|]$ we have

$$|\{\pi_{i,\ell}(a_{i,j}) : \ell \in [d]\}| \;=\; \min\{d, |B_i|\}\,. \tag{2.3}$$

Let $\pi_\ell : [n] \to [n]$ be the bijection given by taking the "disjoint union" of $\pi_{1,\ell}, \ldots, \pi_{t,\ell}$. We claim that $\mathcal{A}_d(f) = \{\pi_1, \ldots, \pi_d\}$ satisfies the conditions of the lemma.

It suffices to verify that this choice of $\mathcal{A}_d(f)$ d-covers f, i.e., to verify that every $r \in [n]$ satisfies at least one of the two conditions in Definition 2.3. Pick any $r \in [n]$. Suppose $r \in A_i$, so that $f(r) \in B_i$ and $\pi_\ell(r) = \pi_{i,\ell}(r)$. If $|B_i| > d$, then $|f^{-1}(f(r))| = |A_i| = |B_i| > d$, so condition (ii) holds. Otherwise, from Eq. (2.3), we conclude that $\{\pi_{i,\ell}(r) : \ell \in [d]\} = B_i$. Therefore, for each $s \in B_i$ — in particular, for $s = f(r)$ — there exists an $\ell \in [d]$ such that $\pi_\ell(r) = \pi_{i,\ell}(r) = s$, so condition (i) holds. ∎

Proof of Lemma 2.2. Let $(i, \pi, x) \in [n] \times \mathcal{S}_n \times \{0,1\}^n$ denote an input for the problem $\text{MPJ}_3^{\text{perm}}$. Then the desired output is $x_{\pi(i)}$. The existence of a protocol P for $\text{MPJ}_3^{\text{perm}}$ with $\text{cost}(P) = O(n\phi(n))$ means that there exist functions

$$\alpha : \mathcal{S}_n \times \{0,1\}^n \to \{0,1\}^m\,,\ \beta : [n] \times \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^m\,,\ \text{and}$$
$$\gamma : [n] \times \mathcal{S}_n \times \{0,1\}^m \times \{0,1\}^m \to \{0,1\}\,,$$

where $m = O(n\phi(n))$, such that $\gamma(i, \pi, \alpha(\pi, x), \beta(i, x, \alpha(\pi, x))) = x_{\pi(i)}$. The functions α, β and γ yield the messages in P of $\text{PLR}_1, \text{PLR}_2$ and PLR_3 respectively.

To design a protocol for MPJ_3, we first let PLR_1 and PLR_3 agree on a parameter d, to be fixed below, and a choice of $\mathcal{A}_d(f)$ for each $f : [n] \to [n]$, as guaranteed by Lemma 2.4. Now, let $(i, f, x) \in [n] \times [n]^{[n]} \times \{0,1\}^n$ be an input for MPJ_3. Our protocol works as follows.

- PLR_1 sends a two-part message. The first part consists of the strings $\{\alpha(\pi, x)\}_\pi$ for all $\pi \in \mathcal{A}_d(f)$. The second part consists of the bits x_s for $s \in [n]$ such that $|f^{-1}(s)| > d$.
- PLR_2 sends the strings $\{\beta(i, x, \alpha)\}_\alpha$ for all strings α in the first part of PLR_1's message.
- PLR_3 can now output $x_{f(i)}$ as follows. If $|f^{-1}(f(i))| > d$, then she reads $x_{f(i)}$ off from the second part of PLR_1's message. Otherwise, since $\mathcal{A}_d(f)$ d-covers f, there exists a $\pi_0 \in \mathcal{A}_d(f)$ such that $f(i) = \pi_0(i)$. She uses the string $\alpha_0 := \alpha(\pi_0, x)$ from

the first part of PLR_1's message and the string $\beta_0 := \beta(i, x, \alpha_0)$ from PLR_2's message to output $\gamma(i, \pi_0, \alpha_0, \beta_0)$.

To verify correctness, we only need to check that PLR_3's output in the "otherwise" case indeed equals $x_{f(i)}$. By the correctness of P, the output equals $x_{\pi_0(i)}$ and we are done, since $f(i) = \pi_0(i)$.

We now turn to the communication cost of the protocol. By the guarantees in Lemma 2.4, $|\mathcal{A}_d(f)| \leq d$, so the first part of PLR_1's message is at most dm bits long, as is PLR_2's message. Since there can be at most n/d values $s \in [n]$ such that $|f^{-1}(s)| > d$, the second part of PLR_2's message is at most n/d bits long. Therefore the communication cost is at most $2dm + n/d = O(dn\phi(n) + n/d)$. Setting $d = \lceil 1/\sqrt{\phi(n)} \rceil$ gives us a bound of $O(n\sqrt{\phi(n)})$, as desired. ∎

2.3. A k-Player Protocol

We now show how to prove Theorem 1.3 by generalizing the protocol from Lemma 2.2 into a protocol for k players. It will help to view an instance of MPJ_k as incorporating several "embedded" instances of MPJ_3. The following lemma makes this precise.

Lemma 2.5. *Let $(i, f_2, \ldots, f_{k-1}, x)$ be input for MPJ_k. Then, for all $1 < j < k$,*

$$\text{MPJ}_k(i, f_2, \ldots, x) = \text{MPJ}_3(f_{j-1} \circ \cdots \circ f_2(i), f_j, x \circ f_{k-1} \circ \cdots \circ f_{j+1}).$$

In our protocol for MPJ_k, for $2 \leq j \leq k-1$, the players $\text{PLR}_1, \text{PLR}_j$, and PLR_k will use a modified version of the protocol from Lemma 2.2 for MPJ_3 on input $(f_{j-1} \circ \cdots \circ f_2(i), f_j, x \circ \cdots \circ f_{j+1})$. Before we get to the protocol, we need to generalize the technical definition and lemma from the previous subsection.

Definition 2.6. Let $S \subseteq [n]$ and let d be a positive integer. A set $\mathcal{A} \subseteq \mathcal{S}_n$ of permutations is said to (S, d)-cover a function $f : [n] \to [n]$ if, for each $r \in S$, at least one of the following conditions holds:

(i) $\exists\, \pi \in \mathcal{A}$ such that $\pi(r) = f(r)$, or
(ii) $|S \cap f^{-1}(f(r))| > d$.

Lemma 2.7. *Let $f : [n] \to [n]$ be a function, $S \subseteq [n]$, and d be a positive integer. There exists a set $\mathcal{A}_{S,d}(f) \subseteq \mathcal{S}_n$, with $|\mathcal{A}_{S,d}(f)| \leq d$, that (S, d)-covers f.*

Proof. This proof closely follows that of Lemma 2.4. As before, we give an explicit algorithm to construct $\mathcal{A}_{S,d}(f)$. Suppose $\text{Range}(f) = \{s_1, s_2, \ldots, s_t\}$, and let $\{A_i\}$ and $\{B_i\}$ be defined as in Lemma 2.4. Let $a_{i,1} < \cdots < a_{i,z}$ be the elements of $A_i \cap S$ arranged in ascending order, and let $a_{i,z+1} < \cdots < a_{i,|A_i|}$ be the elements of $A_i \setminus S$ arranged in ascending order. Similarly, let $b_{i,1} < \cdots < b_{i,|B_i|-1}$ be the elements of $B_i \setminus \{s_i\}$ arranged in ascending order, and let $b_{i,|B_i|} = s_i$. For $i \in [t], \ell \in [d]$, we define $\pi_{i,\ell}(a_{i,j}) := b_{i,(j-\ell) \bmod |B_i|}$. As before, it is routine to verify that $\pi_{i,\ell}$ is a bijection. Let $\pi_\ell : [n] \to [n]$ be the bijection given by taking the "disjoint union" of $\pi_{1,\ell}, \ldots, \pi_{t,\ell}$. We claim that $\mathcal{A}_{S,d}(f) = \{\pi_1, \ldots, \pi_d\}$ satisfies the conditions of the lemma.

It suffices to verify that this choice of $\mathcal{A}_{S,d}(f)$ (S, d)-covers f, i.e., to verify that every $r \in S$ satisfies at least one of the two conditions in Definition 2.6. Pick any $r \in S$. Suppose $r \in A_i$, and fix j such that $r = a_{i,j}$. If $|S \cap f^{-1}(f(r))| > d$, then condition (ii) holds. Otherwise, setting $\ell = j < |S \cap f^{-1}(f(i))| \leq d$, we conclude that $\pi_\ell(r) = \pi_{i,\ell}(r) = \pi_{i,\ell}(a_{i,j}) = b_{i,|B_i|} = s_i = f(r)$, so condition (i) holds. ∎

Proof of Theorem 1.3. To design a protocol for MPJ_k, we first let PLR_1 and PLR_k agree on a parameter d, to be fixed below. They also agree on a choice of $\mathcal{A}_{S,d}(f)$ for all $S \subseteq [n]$ and $f : [n] \to [n]$. Let $(i, f_2, \ldots, f_{k-1}, x)$ denote an input for MPJ_k. Also, let $S_1 = [n]$, and for all $2 \le j \le k-1$, let $S_j = \{s \in [n] : |S_{j-1} \cap f_j^{-1}(s)| > d\}$. Our protocol works as follows:

- PLR_1 sends a $(k-1)$-part message. For $1 \le j \le k-2$, the jth part of PLR_1's message consists of the strings $\{\alpha(\pi, \hat{x}_{j+1})\}_\pi$ for each $\pi \in \mathcal{A}_{S_j,d}(f_{j+1})$. The remaining part consists of the bits x_s for $s \in S_{k-1}$.
- For $2 \le j \le k-1$, PLR_j sends the strings $\{\beta(\hat{i}_j, \hat{x}_j, \alpha)\}_\alpha$ for all strings α in the $(j-1)$th part of PLR_1's message.
- PLR_k can now output $x_{\hat{i}_k}$ as follows. If $|S_1 \cap f_2^{-1}(f_2(i))| \le d$, then, because $\mathcal{A}_{S_1,d}(f_2)$ (S_1, d)-covers f_2, there exists $\pi_0 \in \mathcal{A}_{S_1,d}(f_2)$ such that $f_2(i) = \pi_0(i)$. She uses the string $\alpha_0 = \alpha(\pi_0, \hat{x}_2)$ from the first part of PLR_1's message and the string $\beta_0 = \beta(i, \hat{x}_2, \alpha_0)$ from PLR_2's message to output $\gamma_0 = \gamma(i, \pi_0, \alpha_0, \beta_0)$. Similarly, if there is a j such that $2 \le j \le k-2$ and $|S_j \cap f_{j+1}^{-1}(f_{j+1}(\hat{i}_{j+1}))| \le d$, then since $\mathcal{A}_{S_j,d}(f_{j+1})$ (S_j, d)-covers f_{j+1}, there exists a $\pi_0 \in \mathcal{A}_{S_j,d}(f_{j+1})$ such that $f_{j+1}(\hat{i}_{j+1}) = \pi_0(\hat{i}_{j+1})$. She uses the string $\alpha_0 = \alpha(\pi_0, \hat{x}_{j+1})$ from the jth part of PLR_1's message and the string $\beta_0 = \beta(\hat{i}_{j+1}, \hat{x}_{j+1}, \alpha_0)$ from PLR_{j+1}'s message to output $\gamma_0 = \gamma(\hat{i}_{j+1}, \pi_0, \alpha_0, \beta_0)$. Otherwise, $|S_{k-2} \cap f_{k-1}^{-1}(f_{k-1}(\hat{i}_{k-1}))| > d$, hence $\hat{i}_k \in S_{k-1}$, and she reads $x_{\hat{i}_k}$ off from the last part of PLR_1's message.

To verify correctness, we need to ensure that PLR_k always outputs $x \circ f_{k-1} \circ \cdots \circ f_2(i)$. In the following argument, we repeatedly use Lemma 2.5. We proceed inductively. If $|S_1 \cap f_2^{-1}(f_2(i))| \le d$ then there exists $\pi_0 \in \mathcal{A}_{S_1,d}(f_2)$ such that $f_2(i) = \pi_0(i)$, $\alpha_0 = \alpha(\pi_0, \hat{x}_2)$, and $\beta_0 = \beta(i, \hat{x}_2, \alpha_0)$, and PLR_k outputs $\gamma_0 = \gamma(i, \pi_0, \alpha_0, \beta_0) = \hat{x}_2(\pi_0(i)) = x \circ f_{k-1} \circ \cdots \circ f_2(i)$. Otherwise, $|S_1 \cap f_2^{-1}(f_2(i))| > d$, hence $f_2(i) \in S_2$. Inductively, if $\hat{i}_j \in S_{j-1}$, then either $|S_{j-1} \cap f_j^{-1}(f_j(\hat{i}_j))| \le d$, or $|S_{j-1} \cap f_j^{-1}(f_j(\hat{i}_j))| > d$. In the former case, there is $\pi_0 \in \mathcal{A}_{S_{j-1},d}(f_j)$ such that $f_j(\hat{i}_j) = \pi_0(\hat{i}_j)$; $\alpha_0(\pi_0, \hat{x}_j)$, and $\beta_0 = \beta(\hat{i}_j, \hat{x}_j, \alpha_0)$, and PLR_k outputs $\gamma_0 = \gamma(\hat{i}_j, \pi_0, \alpha_0, \beta_0) = \hat{x}_j(f_j(\hat{i}_j)) = x \circ f_{k-1} \circ \cdots \circ f_2(i)$. In the latter case, $f_j(\hat{i}_j) \in S_j$. By induction, we have that either PLR_k outputs $x \circ f_{k-1} \circ \cdots \circ f_2(i)$, or $\hat{i}_k \in S_{k-1}$. But in this case, PLR_k outputs $x(\hat{i}_k) = x \circ f_{k-1} \circ \cdots \circ f_2(i)$ directly from the last part of PLR_1's message. Therefore, PLR_k always outputs $x \circ f_{k-1} \circ \cdots \circ f_2(i)$ correctly.

We now turn to the communication cost of the protocol. By Lemma 2.7, $|\mathcal{A}_{S_j,d}(f_j)| \le d$ for each $2 \le j \le k-1$, hence the first $k-2$ parts of PLR_1's message each are at most dm bits long, as is PLR_j's message for all $2 \le j \le k-1$. Also, since for all $2 \le j \le k-1$, there are at most $|S_{j-1}|/d$ elements $s \in S_j$ such that $|S_{j-1} \cap f_j^{-1}(s)| > d$, we must have that $|S_2| \le |S_1|/d = n/d, |S_3| \le |S_2|/d \le n/d^2$, etc., and $|S_{k-1}| \le n/d^{k-2}$. Therefore, the final part of PLR_1's message is at most n/d^{k-2} bits long, and the total communication cost is at most $2(k-2)dm + n/d^{k-2} = O((k-2)dn\phi(n) + n/d^{k-2})$. Setting $d = \lceil 1/((k-2)\phi(n))^{1/(k-1)} \rceil$ gives us a bound of $O(n(k\phi(n))^{(k-2)/(k-1)})$ as desired. ∎

Note that, in the above protocol, except for the first and last players, the remaining players access very limited information about their input. Specifically, for all $2 \le j \le k-1$, PLR_j needs to see only $\hat{i}_j$ and $\hat{x}_j$, i.e., PLR_j is both *conservative* and *collapsing*. Despite this severe restriction, we have a sublinear protocol for MPJ_k. As we shall see in the next section, further restricting the input such that PLR_1 is also collapsing yields very strong lower bounds.

3. Collapsing Protocols: A Lower Bound

Let $F : \mathscr{A}_1 \times \mathscr{A}_2 \times \cdots \times \mathscr{A}_k \to \mathscr{B}$ be a k-player NOF communication problem and P be a protocol for F. We say that PLR_j is *collapsing* in P if her message depends only on $x_1, \ldots, x_{j-1}$ and the function $g_{x,j} : \mathscr{A}_1 \times \mathscr{A}_2 \times \cdots \times \mathscr{A}_j \to \mathscr{B}$ given by $g_{x,j}(z_1, \ldots, z_j) = F(z_1, \ldots, z_j, x_{j+1}, \ldots, x_k)$. For pointer jumping, this amounts to saying that PLR_j sees all layers $1, \ldots, j-1$ of edges (i.e., the layers preceding the one on her forehead), but not layers $j+1, \ldots, k$; however, she does see the result of following the pointers from each vertex in layer j. Still more precisely, if the input to MPJ_k (or $\widehat{\text{MPJ}}_k$) is $(i, f_2, \ldots, f_k)$, then the only information PLR_j gets is $i, f_2, \ldots, f_{j-1}$ and the composition $f_k \circ f_{k-1} \circ \cdots \circ f_{j+1}$.

We say that a protocol is collapsing if every player involved is collapsing. We shall prove Theorem 1.4 by contradiction. Assume that there is a collapsing protocol P for MPJ_k in which every player sends less than $n - \frac{1}{2}\log n - 2$ bits. We shall construct a pair of inputs that differ only in the last layer (i.e., the Boolean string on PLR_k's forehead) and that cause players 1 through $k-1$ to send the exact same sequence of messages. This will cause PLR_k to give the same output for both these inputs. But our construction will ensure that the desired outputs are unequal, a contradiction. To aid our construction, we need some definitions and preliminary lemmas.

Definition 3.1. A string $x \in \{0,1\}^n$ is said to be *consistent* with $(f_1, \ldots, f_j, \alpha_1, \ldots, \alpha_j)$ if, in protocol P, for all $h \le j$, PLR_h sends the message α_h on seeing input $(i = f_1, \ldots, f_{h-1}, x \circ f_j \circ f_{j-1} \circ \cdots \circ f_{h+1})$ and previous messages $\alpha_1, \ldots, \alpha_{h-1}$.[2] A subset $T \subseteq \{0,1\}^n$ is said to be consistent with $(f_1, \ldots, f_j, \alpha_1, \ldots, \alpha_j)$ if x is consistent with $(f_1, \ldots, f_j, \alpha_1, \ldots, \alpha_j)$ for all $x \in T$.

Definition 3.2. For strings $x, x' \in \{0,1\}^n$ and $a, b \in \{0,1\}$, define the sets

$$I_{ab}(x, x') \;:=\; \{j \in [n] : (x_j, x'_j) = (a, b)\}\,.$$

A pair of strings (x, x') is said to be a *crossing pair* if for all $a, b \in \{0,1\}$, $I_{ab}(x, x') \neq \emptyset$. A set $T \subseteq \{0,1\}^n$ is said to be *crossed* if it contains a crossing pair and *uncrossed* otherwise. The *weight* of a string $x \in \{0,1\}^n$ is defined to be the number of 1s in x, and denoted $|x|$.

For the rest of this section, we assume (without loss of generality) that n is large enough and even.

Lemma 3.3. *If $T \subseteq \{0,1\}^n$ is uncrossed, then $|\{x \in T : |x| = n/2\}| \le 2$.*

Proof. Let x and x' be distinct elements of T with $|x| = |x'| = n/2$. For $a, b \in \{0,1\}$, define $t_{ab} = |I_{ab}(x, x')|$. Since $x \neq x'$, we must have $t_{01} + t_{10} > 0$. An easy counting argument shows that $t_{01} = t_{10}$ and $t_{00} = t_{11}$. Since T is uncrossed, (x, x') is not a crossing pair, so at least one of the numbers t_{ab} must be zero. It follows that $t_{00} = t_{11} = 0$, so x and x' are bitwise complements of each other. Since this holds for any two strings in $\{x \in T : |x| = n/2\}$, that set can have size at most 2. ∎

Lemma 3.4. *Suppose $t \le n - \frac{1}{2}\log n - 2$. If $\{0,1\}^n$ is partitioned into 2^t disjoint sets, then one of those sets must be crossed.*

[2] It is worth noting that, in Definition 3.1, x is not to be thought of as an input on PLR_k's forehead. Instead, in general, it is the composition of the rightmost $k - j$ layers of the input graph.

Proof. Let $\{0,1\}^n = T_1 \sqcup T_2 \sqcup \cdots \sqcup T_m$ be a partition of $\{0,1\}^n$ into m uncrossed sets. Define $X := \{x \in \{0,1\}^n : |x| = n/2\}$. Then $X = \bigcup_{i=1}^m (T_i \cap X)$. By Lemma 3.3,

$$|X| \;\leq\; \sum_{i=1}^{m} |T_i \cap X| \;\leq\; 2m\,.$$

Using Stirling's approximation, we can bound $|X| > 2^n/(2\sqrt{n})$. Therefore, $m > 2^{n-\frac{1}{2}\log n - 2}$. ∎

Proof of Theorem 1.4. Set $t = n - \frac{1}{2}\log n - 2$. Recall that we have assumed that there is a collapsing protocol P for MPJ_k in which every player sends at most t bits. We shall prove the following statement by induction on j, for $j \in [k-1]$.

(*) There exists a partial input $(i = f_1, f_2, \ldots, f_j) \in [n] \times \left([n]^{[n]}\right)^{j-1}$, a sequence of messages $(\alpha_1, \ldots, \alpha_j)$ and a crossing pair of strings $(x, x') \in (\{0,1\}^n)^2$ such that both x and x' are consistent with $(f_1, \ldots, f_j, \alpha_1, \ldots, \alpha_j)$, whereas $x \circ f_j \circ \cdots \circ f_2(i) = 0$ and $x' \circ f_j \circ \cdots \circ f_2(i) = 1$.

Considering (*) for $j = k-1$, we see that PLR_k must behave identically on the two inputs $(i, f_2, \ldots, f_{k-1}, x)$ and $(i, f_2, \ldots, f_{k-1}, x')$. Therefore, she must err on one of these two inputs. This will give us the desired contradiction.

To prove (*) for $j = 1$, note that PLR_1's message, being at most t bits long, partitions $\{0,1\}^n$ into at most 2^t disjoint sets. By Lemma 3.4, one of these sets, say T, must be crossed. Let (x, x') be a crossing pair in T and let α_1 be the message that PLR_1 sends on seeing a string in T. Fix $i = f_1$ such that $i \in I_{01}(x, x')$. These choices are easily seen to satisfy the conditions in (*). Now, suppose (*) holds for a particular $j \geq 1$. Fix the partial input $(f_1, \ldots, f_j)$ and the message sequence $(\alpha_1, \ldots, \alpha_j)$ as given by (*). We shall come up with appropriate choices for f_{j+1}, α_{j+1} and a new crossing pair (y, y') to replace (x, x'), so that (*) is satisfied for $j+1$. Since PLR_{j+1} sends at most t bits, she partitions $\{0,1\}^n$ into at most 2^t subsets (the partition might depend on the choice of $(f_1, \ldots, f_j, \alpha_1, \ldots, \alpha_j)$).

As above, by Lemma 3.4, she sends a message α_{j+1} on some crossing pair (y, y'). Choose f_{j+1} so that it maps $I_{ab}(x, x')$ to $I_{ab}(y, y')$ for all $a, b \in \{0,1\}$; this is possible because $I_{ab}(y, y') \neq \emptyset$. Then, for all $i \in [n]$, $x_i = y_{f_{j+1}(i)}$ and $x'_i = y'_{f_{j+1}(i)}$. Hence, $x = y \circ f_{j+1}$ and $x' = y' \circ f_{j+1}$. Applying the inductive hypothesis and the definition of consistency, it is straightforward to verify the conditions of (*) with these choices for f_{j+1}, α_{j+1}, y and y'. This completes the proof. ∎

4. Collapsing Protocols: An Upper Bound

We now turn to proving Theorem 1.5 by constructing an appropriate collapsing protocol for $\widehat{\text{MPJ}}_k^{\text{perm}}$. Our protocol uses what we call *bucketing schemes*, which have the flavor of the conservative protocol of Damm et al. [DJS98]. For any function $f \in [n]^{[n]}$ and any $S \subseteq [n]$, let $\mathbf{1}_S$ denote the indicator function for S; that is, $\mathbf{1}_S(i) = 1 \Leftrightarrow i \in S$. Also, let $f|_S$ denote the function f restricted to S; this can be seen as a list of numbers $\{i_s\}$, one for each $s \in S$. Players will often need to send $\mathbf{1}_S$ and $f|_S$ together in a single message. This is because later players might not know S, and will therefore be unable to interpret $f|_S$ without $\mathbf{1}_S$. Let $\langle m_1, \ldots, m_t \rangle$ denote the concatenation of messages $m_1, \ldots, m_t$.

Definition 4.1. A *bucketing scheme* on a set X is an ordered partition $\mathcal{B} = (B_1, \ldots, B_t)$ of X into *buckets*. For $x \in X$, we write $\mathcal{B}[x]$ to denote the unique integer j such that $B_j \ni x$.

We actually prove our upper bound for problems slightly more general than $\widehat{\text{MPJ}}_k^{\text{perm}}$. To be precise, for an instance $(i, f_2, \ldots, f_k)$ of $\widehat{\text{MPJ}}_k$, we allow any one of $f_2, \ldots, f_k$ to be an arbitrary function in $[n]^{[n]}$. The rest of the f_js are required to be permutations, i.e., in $\mathcal{S}_n$.

Theorem 4.2 (Slight generalization of Theorem 1.5)**.** *There is an $O(n \log^{(k-1)} n)$ collapsing protocol for instance $(i, f_2, \ldots, f_k)$ of $\widehat{\text{MPJ}}_k$ when all but one of $f_2, \ldots, f_k$ are permutations. In particular, there is such a protocol for $\widehat{\text{MPJ}}_k^{\text{perm}}$.*

Proof. We prove this for $\widehat{\text{MPJ}}_k^{\text{perm}}$ only. For $1 \le t \le \lceil \log n \rceil$, define the bucketing scheme $\mathcal{B}_t = (B_1, \ldots, B_{2^t})$ on $[n]$ by $B_j := \{r \in [n] : \lceil 2^t r/n \rceil = j\}$. Note that each $|B_j| \le \lceil n/2^t \rceil$ and that a bucket can be described using t bits. For $1 \le j \le k$, let $b_j = \lceil \log^{(k-j)} n \rceil$. In the protocol, most players will use two bucketing schemes, $\mathcal{B}$ and $\mathcal{B}'$. On input $(i, f_2, \ldots, f_k)$:

- PLR_1 sees $\hat{f}_1$, computes $\mathcal{B}' := \mathcal{B}_{b_1}$, and sends $\langle \mathcal{B}'[\hat{f}_1(1)], \ldots, \mathcal{B}'[\hat{f}_1(n)] \rangle$.
- PLR_2 sees $\hat{i}_2, \hat{f}_2$, and PLR_1's message. PLR_2 computes $\mathcal{B} := \mathcal{B}_{b_1}$ and $\mathcal{B}' := \mathcal{B}_{b_2}$. She recovers $b := \mathcal{B}[\hat{f}_2(f_2(\hat{i}_2))]$ and hence B_b. Let $S_2 := \{s \in [n] : \hat{f}_2(s) \in B_b\}$. Note that $f_2(\hat{i}_2) \in S_2$. PLR_2 sends $\langle \mathbf{1}_{S_2}, \{\mathcal{B}'[\hat{f}_2(s)] : s \in S_2\} \rangle$.

$$\vdots$$

- PLR_j sees $\hat{i}_j, \hat{f}_j$, and PLR_{j-1}'s message. PLR_j computes $\mathcal{B} := \mathcal{B}_{b_{j-1}}$ and $\mathcal{B}' := \mathcal{B}_{b_j}$. She recovers $b := \mathcal{B}[\hat{f}_j(f_j(\hat{i}_j))]$ and hence B_b. Let $S_j := \{s \in [n] : \hat{f}_j(s) \in B_b\}$. Note that the definitions guarantee that $f_j(\hat{i}_j) \in S_j$. PLR_j sends $\langle \mathbf{1}_{S_j}, \{\mathcal{B}'[\hat{f}_j(s)] : s \in S_j\} \rangle$.

$$\vdots$$

- PLR_k sees $\hat{i}_k$ and PLR_{k-1}'s message and outputs $f_k(\hat{i}_k)$.

We claim that this protocol costs $O(n \log^{(k-1)} n)$ and correctly outputs $\widehat{\text{MPJ}}_k(i, f_2, \ldots, f_k)$. For each $2 \le j \le k-1$, PLR_j uses bucketing scheme $\mathcal{B}_{b_{j-1}}$ to recover the bucket B_b containing $\hat{f}_j(f_j(\hat{i}_j))$. She then encodes each element in B_b in the bucketing scheme $\mathcal{B}_{b_j}$. Each bucket in $\mathcal{B}_{b_j}$ has size at most $\lceil n/b_{j+1} \rceil$. In particular, each bucket in scheme $\mathcal{B}_{k-1}$ has size at most $\lceil n/b_k \rceil = 1$, and the unique element in the bucket (if present) is precisely $f_k(\hat{i}_k)$. Turning to the communication cost, PLR_1 sends $b_1 = \lceil \log^{(k-1)} n \rceil$ bits to identify the bucket for each $i \in [n]$, giving a total of $n\lceil \log^{(k-1)} n \rceil$ bits. For $1 < j < k$, PLR_j uses $n + b_j(n/b_j) = O(n)$ bits. Thus, the total cost is $O(n \log^{(k-1)} n + kn)$ bits.

For $k \le \log^* n$ players, we are done. For larger k, we can get an $O(n)$ protocol by doubling the size of each b_j and stopping the protocol when the buckets have size ≤ 1. ∎

5. Concluding Remarks

We have presented the first nontrivial upper bound on the NOF communication complexity of the Boolean problem MPJ_k, showing that $C(\text{MPJ}_k) = o(n)$. A lower bound of $\Omega(n)$ had seemed *a priori* reasonable, but we show that this is not the case. One plausible line of attack on lower bounds for MPJ_k is to treat it as a *direct sum* problem: at each player's turn, it seems that n different paths need to be followed in the input graph, so it seems that an information theoretic approach (as in Bar-Yossef et al. [BJKS02] or Chakrabarti [Cha07]) could lower bound $C(\text{MPJ}_k)$ by n times the complexity of some simpler problem. However, it appears that such an approach would naturally yield a lower bound of the form $\Omega(n/\xi(k))$, as in Conjecture 1.1, which we have explicitly falsified.

The most outstanding open problem regarding MPJ_k is to resolve Conjecture 1.2. A less ambitious, but seemingly difficult, goal is to get tight bounds on $C(\text{MPJ}_3)$, closing the gap between our $O(n\sqrt{\log\log n/\log n})$ upper bound and Wigderson's $\Omega(\sqrt{n})$ lower bound. A still less ambitious question is prove that MPJ_3 is harder than its very special subproblem TPJ_3 (defined in Section 1.1). Our $n - O(\log n)$ lower bound for collapsing protocols is a step in the direction of improving the known lower bounds. We hope our technique provides some insight about the more general problem.

References

[AB07] Sanjeev Arora and Boaz Barak. *Complexity Theory: A Modern Approach*. Available online at ⟨http://www.cs.princeton.edu/theory/complexity/⟩, 2007.

[Abl96] Farid Ablayev. Lower bounds for one-way probabilistic communication complexity and their application to space complexity. *Theoretical Computer Science*, 175(2):139–159, 1996.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. Preliminary version in *Proc. 28th Annu. ACM Symp. Theory Comput.*., pages 20–29, 1996.

[BHK01] László Babai, Thomas P. Hayes, and Peter G. Kimmel. The cost of the missing bit: Communication complexity with help. *Combinatorica*, 21(4):455–488, 2001.

[BJKS02] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 209–218, 2002.

[BPS05] Paul Beame, Toniann Pitassi, and Nathan Segerlind. Lower bounds for Lovász-Schrijver systems and beyond follow from multiparty communication complexity. In *Proc. 32nd International Colloquium on Automata, Languages and Programming*, pages 1176–1188, 2005.

[BT94] Richard Beigel and Jun Tarui. On ACC. *Comput. Complexity*, 4:350–366, 1994.

[CFL83] Ashok K. Chandra, Merrick L. Furst, and Richard J. Lipton. Multi-party protocols. In *Proc. 15th Annual ACM Symposium on the Theory of Computing*, pages 94–99, 1983.

[Cha07] Amit Chakrabarti. Lower bounds for multi-player pointer jumping. In *Proc. 22nd Annual IEEE Conference on Computational Complexity*, pages 33–45, 2007.

[CJP08] Amit Chakrabarti, T. S. Jayram, and Mihai Pătraşcu. Tight lower bounds for selection in randomly ordered streams. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2008. to appear.

[DJS98] Carsten Damm, Stasys Jukna, and Jiří Sgall. Some bounds on multiparty communication complexity of pointer jumping. *Comput. Complexity*, 7(2):109–127, 1998. Preliminary version in *Proc. 13th International Symposium on Theoretical Aspects of Computer Science*, pages 643–654, 1996.

[GM07] Sudipto Guha and Andrew McGregor. Lower bounds for quantile estimation in random-order and multi-pass streaming. In *Proc. 34th International Colloquium on Automata, Languages and Programming*, pages 704–715, 2007.

[HG91] Johan Håstad and Mikael Goldmann. On the power of small-depth threshold circuits. *Comput. Complexity*, 1:113–129, 1991.

[NW93] Noam Nisan and Avi Wigderson. Rounds in communication complexity revisited. *SICOMP*, 22(1):211–219, 1993. Preliminary version in *Proc. 23rd Annu. ACM Symp. Theory Comput.*, pages 419–429, 1991.

[PRS97] Pavel Pudlák, Vojtěch Rödl, and Jiří Sgall. Boolean circuits, tensor ranks and communication complexity. *SIAM J. Comput.*, 26(3):605–633, 1997.

[VW07] Emanuele Viola and Avi Wigderson. One-way multi-party communication lower bound for pointer jumping with applications. In *Proc. 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 427–437, 2007.

[Yao90] Andrew C. Yao. On ACC and threshold circuits. In *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 619–627, 1990.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 157-168
www.stacs-conf.org

FINDING IRREFUTABLE CERTIFICATES FOR S_2^p VIA ARTHUR AND MERLIN

VENKATESAN T. CHAKARAVARTHY AND SAMBUDDHA ROY

IBM India Research Lab, New Delhi.
E-mail address: {vechakra,sambuddha}@in.ibm.com

Abstract. We show that $S_2^p \subseteq P^{prAM}$, where S_2^p is the symmetric alternation class and prAM refers to the promise version of the Arthur-Merlin class AM. This is derived as a consequence of our main result that presents an FP^{prAM} algorithm for finding a small set of "collectively irrefutable certificates" of a given S_2-type matrix. The main result also yields some new consequences of the hypothesis that NP has polynomial size circuits. It is known that the above hypothesis implies a collapse of the polynomial time hierarchy (PH) to $S_2^p \subseteq ZPP^{NP}$ [5, 14]. Under the same hypothesis, we show that PH collapses to P^{prMA}. We also describe an FP^{prMA} algorithm for learning polynomial size circuits for SAT, assuming such circuits exist. For the same problem, the previously best known result was a ZPP^{NP} algorithm [4].

1. Introduction

We consider the problem of finding irrefutable certificates for the symmetric alternation class S_2^p. The class S_2^p was introduced by Russell and Sundaram [17] and independently, by Canetti [6]. A language L in the class S_2^p is characterized by an interactive proof system of the following type. The proof system consists of two computationally all-powerful provers called the Yes-prover and the No-prover, and a polynomial time verifier. The verifier interacts with the two provers to ascertain whether or not an input string x belongs to the language L. The Yes-prover and the No-prover make contradictory claims: $x \in L$ and $x \notin L$, respectively. Of course, only one of them is honest. To substantiate their claims, the provers provide strings y and z as certificates. The verifier analyzes the input x and the two certificates and votes in favor of one of the provers. If the Yes-prover wins the vote, we say that y *beats* z and we say that z *beats* y, otherwise. The requirement is that, if $x \in L$, then the Yes-prover must have a certificate y that beats any certificate z given by the No-prover. Similarly, if $x \notin L$, the No-prover must have a certificate z that beats any certificate y given by the Yes-prover. We call certificates satisfying the above requirements as *irrefutable certificates* (written IC). Clearly, for any input string, only the honest prover has an IC.

Cai [5] showed that $S_2^p \subseteq ZPP^{NP}$. Let us rephrase this result: for any language $L \in S_2^p$, we have a ZPP^{NP} algorithm that takes an input string and decides whether the Yes-prover

Key words and phrases: Symmetric alternation, promise-AM, Karp–Lipton theorem, learning circuits.

has an IC or the NO-PROVER has an IC. The main purpose of this paper is to study the problem of *finding* IC's for an input string.

The above problem and the related issues regarding S_2^p can conveniently be described in terms of Boolean matrices. Let L be a language in S_2^p and x be an input string. Let n and m, denote the length of the certificates of the YES-PROVER and NO-PROVER, respectively. We model the behaviour of the verifier on the input x in the form of a $2^n \times 2^m$ Boolean matrix M. In the matrix M, the rows correspond to the certificates of the YES-PROVER and the columns correspond to the certificates of the NO-PROVER. For certificates $y \in \{0,1\}^n$ and $z \in \{0,1\}^m$, if y beats z, then we set $M[y,z] = 1$ and if z beats y, then we set $M[y,z] = 0$. Notice that the matrix M has either a row full of 1's or a column full of 0's. The first scenario happens, when $x \in L$ (here, the row full of 1's corresponds to an IC of the YES-PROVER). Similarly, the second scenario happens, when $x \notin L$ (here, the column full of 0's corresponds to an IC of the NO-PROVER). We call any Boolean matrix satisfying the above condition as an S_2-*type* matrix. A row full of 1's is called a row-side IC and a column full of 0's is called a column-side IC. (Notice that a Boolean matrix cannot have both.) Though the matrix M is exponentially large in the size of the input $|x|$, it can be *succinctly encoded* in the form of a Boolean circuit C having size polynomial in $|x|$. The circuit C takes as input $y \in \{0,1\}^n$ and $z \in \{0,1\}^m$ and outputs $C(y,z) = M[y,z]$. The circuit achieves this by simulating the verifier's algorithm on the input x. Using standard techniques, we can construct the desired circuit C in time polynomial in $|x|$.

Problems regarding S_2^p can now be expressed as problems on S_2-type matrices, presented succinctly in the form of circuits. First, let us consider the basic problem of membership testing for a language $L \in S_2^p$: given a string x, determine whether $x \in L$ or not. This is equivalent to following problem on S_2-type matrices.

MEMBERSHIP TESTING. Given an S_2-type matrix M, presented succinctly in the form of a circuit, distinguish between the two cases: (i) there exists a row-side IC; (ii) there exists a column-side IC.

Cai [5] showed that $S_2^p \subseteq ZPP^{NP}$. Equivalently, this result presents a ZPP^{NP} algorithm for the MEMBERSHIP TESTING problem. We consider the more general problem of *finding* an IC for a given S_2-type matrix.

Problem FINDIC: Given an S_2-type matrix M, presented succinctly in the form of a circuit, output an IC either on the row side or on the column side.

Via a simple observation, we show that if there exists a ZPP^{NP} algorithm for the FINDIC problem, then the polynomial time hierarchy (PH) collapses. In summary, we can determine in ZPP^{NP} whether an IC is found among the rows or among the columns; but, we cannot find an IC in ZPP^{NP}, unless PH collapses. So, we study the easier problem of finding a set of *collectively irrefutable certificates* (written CIC).

We say that a set of rows Y *collectively beats* a column z, if some row $y \in Y$ beats z. The set Y is said to be a *row-side* CIC, if Y collectively beats every z. The notion of column-side CIC is defined analogously. Notice that an arbitrary Boolean matrix may have both a row-side CIC and a column-side CIC. However, the existence of a row-side CIC precludes there being a column-side IC. Thus, in the case of S_2-type matrices, a row-side CIC shows that there exists a row-side IC (which in turn, means that the input string $x \in L$). Therefore, a row-side CIC is as useful as a row-side IC, in certifying that $x \in L$. Our main result provides an algorithm for finding a CIC of small size (logarithmic in the size of the input matrix).

Problem FINDCIC. Given an S_2-type matrix M of size $2^n \times 2^m$, presented succinctly in the form of a circuit, output either a row-side CIC or a column-side CIC of size $\max\{n, m\}$.

Our main result presents an FP^{prAM} algorithm for the FINDCICproblem, i.e., the algorithm runs in (deterministic) polynomial time making queries to an prAM oracle; prAM refers to the promise version of the Arthur-Merlin class AM.

Main Result. We present an FP^{prAM} algorithm for the FINDCIC problem.

We note that the problem FINDCIC can also be solved by a ZPP^{NP} algorithm; such an algorithm is implicit in the work of Cai [5] and Fortnow et al. [9]. The containment relationships between FP^{prAM} and ZPP^{NP} are not known. This issue is discussed in more detail below.

An immediate corollary of the main result is that $S_2^p \subseteq P^{prAM}$. This gives a nice counterpart to Cai's result [5] that $S_2^p \subseteq ZPP^{NP}$. The containment relationships between P^{prAM} and ZPP^{NP} are unknown. (In fact, it has been a long standing open problem to put AM in Σ_2^p). However, we can show that $P^{prAM} \subseteq BPP^{NP}$. Moreover, Cai's result can also be derived from the main result.

It is known that $P^{NP} \subseteq S_2^p$ [17] and one of the most challenging open problems regarding S_2^p asks whether S_2^p is contained in P^{NP}. Working under a larger framework, Shaltiel and Umans [19] also studied this issue and derived the result $S_2^p = P^{NP}$, under a suitable hardness hypothesis. This was achieved by derandomizing Cai's construction for $S_2^p \subseteq ZPP^{NP}$. The above-mentioned hardness hypothesis was the one used by Miltersen and Vinodchandran [15] to derandomize AM to get AM = NP: there exists a language L in $NE \cap coNE$ so that for all but finitely many n, $L \cap \{0,1\}^n$ has SV-nondeterministic circuit complexity at least $2^{\epsilon n}$. Thus, under the above hypothesis, Shaltiel and Umans showed that $S_2^p = P^{NP}$. Our claim that $S_2^p \subseteq P^{prAM}$ yields an alternative proof of the above result. This is obtained by appealing to the hitting set generator of Miltersen and Vinodchandran [15]. The details will be included in the full version of the paper.

The main result yields two new consequences of the assumption that NP has polynomial size circuits. Under the above assumption, Karp and Lipton [13] showed that the polynomial time hierarchy (PH) collapses to Σ_2^p. Subsequently, their result has been strengthened: Köbler and Watanabe [14] derived the collapse $PH = ZPP^{NP}$; Sengupta observed that $PH = S_2^p \subseteq ZPP^{NP}$ (see [5]); recently, the collapse was improved to $PH = O_2^p \subseteq S_2^p$ [7]. It has been a challenging open problem to get the collapse down to P^{NP}. We derive a weaker result: if NP has polynomial size circuits, then $PH = P^{prMA}$. It is worthwhile to compare this new collapse result with the earlier ones. Though it is known that $P^{MA} \subseteq S_2^p$ [17], it is not clear whether P^{prMA} is contained in S_2^p. However, we can show that $P^{prMA} \subseteq ZPP^{NP}$ (by extending the known result that $MA \subseteq ZPP^{NP}$ [1, 11]).

One implication of the new collapse result is that P^{prMA} cannot have $SIZE(n^k)$ circuits, for any fixed k. However, a stronger result is known: in a recent breakthrough, Santhanam [18] proved the above circuit lowerbound for the class prMA.

In the above context, our next result deals with the problem of learning polynomial size circuits for SAT. Under the assumption that NP has polynomial size circuits, Bshouty et al. [4] designed a ZPP^{NP} algorithm that finds a correct circuit for SAT at a given length. We improve their result by presenting a FP^{prMA} algorithm for the same task.

Finally, we show how to generalize our main result to the case of arbitrary Boolean matrices (that may not necessarily be of S_2-type). For this, we make use of a nice and interesting lemma by Goldreich and Wigderson [10]: they showed that *any* $2^n \times 2^m$ Boolean

matrix M contains a row-side CIC of size m or a column-side CIC of size n (or both). We consider the scenario where the matrix M is presented succinctly in the form of a circuit and describe an $\mathrm{FP}^{\mathrm{prAM}}$ algorithm for finding such a CIC; but, our algorithm suffers a small blow-up in the size of the output CIC. The algorithm finds a row-side CIC of size m^2 or a column-side CIC of size n^2.

For lack of space, the details of the above generalization and proofs for some of the results are omitted in this paper. These will be included in the full version of the paper.

Proof Techniques. The proof of our main result has a flavor similar to that of Cai's result [5]. The proof involves a variant of self-reduction and the tools of approximate counting and testing whether a set is "large" or "small". For the latter two tasks, we borrow ideas from the work of Jerrum et al. [12], Stockmeyer [21] and Sipser [20]. We put together all these ideas and show how to solve our problem using a prAM oracle. Our exposition is largely self-contained.

2. Preliminaries

In this section, we develop definitions and notations used throughout the paper.

Symmetric Alternation. A language L is said to be in the class S_2^p, if there exists a polynomial time computable Boolean predicate $V(\cdot,\cdot,\cdot)$ and polynomials $p(\cdot)$ and $q(\cdot)$ such that for any x, we have

$$\begin{aligned} x \in L &\implies (\exists y \in \{0,1\}^n)(\forall z \in \{0,1\}^m)[V(x,y,z)=1], \text{and} \\ x \notin L &\implies (\exists z \in \{0,1\}^m)(\forall y \in \{0,1\}^n)[V(x,y,z)=0], \end{aligned}$$

where $n = p(|x|)$ and $m = q(|x|)$. We refer to the y's and z's above as *certificates*. The predicate V is called the *verifier*.

Matrix representation of the verifier's computation. Let L be a language in S_2^p via a verifier predicate V. Fix an input string x. It is convenient to represent the behaviour of the verifier on various certificates in the form of a matrix. Define a Boolean $2^n \times 2^m$ matrix M, such that for $y \in \{0,1\}^n$ and $z \in \{0,1\}^m$, $M[y,z] = V(x,y,z)$. Thus, any row or column in M corresponds to a certificate. We call M as the matrix corresponding to the input x. Matrices constructed in the above fashion have some special properties that are derived the from the definition of S_2^p.

S_2-type matrices and irrefutable certificates. Let M be a $2^n \times 2^m$ Boolean matrix. For a row $y \in \{0,1\}^n$ and a column $z \in \{0,1\}^m$, if $M[y,z] = 1$, then y is said to *beat* z; similarly, z is said to beat y, if $M[y,z] = 0$. A row y is called a *row-side* IC, if y beats every column $z \in \{0,1\}^m$; a column z is called a *column-side* IC if z beats every row $y \in \{0,1\}^m$. Notice that a matrix cannot have both a row-side IC and a column-side IC. The matrix M is said to be an S_2-type matrix, if it has either a row-side IC or a column-side IC. A set of rows Y is called a row-side CIC, if for every column z, there exists a row $y \in Y$ such that y beats z. Similarly, a set of columns Z is called a column-side CIC, if for every row y, there exists a column $z \in Z$ such that z beats y.

Remark. Let us put the above discussion in the context of a language $L \in \mathrm{S}_2^p$ and make some simple observations. For any input string x, the matrix M corresponding to x is an S_2-type matrix. The matrix M will have a row-side IC, if and only if $x \in L$; similarly, M will have a column-side IC, if and only if $x \notin L$.

Succinct encoding of matrices and sets. A Boolean circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ is said to *succinctly encode* a Boolean $2^n \times 2^m$ matrix M, if for all $y \in \{0,1\}^n$ and

$z \in \{0,1\}^m$, we have $C(y,z) = M[y,z]$. A Boolean circuit $C : \{0,1\}^m \to \{0,1\}$ is said to *succinctly encode* a set $X \subseteq \{0,1\}^m$, if for all $x \in \{0,1\}^m$, $x \in X \iff C(x) = 1$.

Remark. Let L be a language in S_2^p via a verifier V. Let x be an input string with the corresponding matrix M. Using standard techniques, we can obtain a Boolean circuit $C : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ such that $C(y,z) = V(x,y,z)$. Given the input x, the above task can be performed in time polynomial in $|x|$. The size of the circuit is also polynomial in $|x|$. Notice that the above circuit C succinctly encodes the matrix M.

Complexity classes. We use standard definitions for complexity classes such as P, NP, P/poly, MA, AM, $\mathrm{ZPP}^{\mathrm{NP}}$ and $\mathrm{BPP}^{\mathrm{NP}}$ [8, 16]. Below, we present definitions for promise and function classes, that are central to our paper.

Promise languages. A promise language Π is a pair (Π_1, Π_2), where $\Pi_1, \Pi_2 \subseteq \Sigma^*$, such that $\Pi_1 \cap \Pi_2 = \emptyset$. The elements of Π_1 are called the *positive instances* and those of Π_2 are called the *negative instances*.

Promise MA (prMA). A promise language $\Pi = (\Pi_1, \Pi_2)$ is said to be in the promise class prMA, if there exists a polynomial time computable Boolean predicate $A(\cdot,\cdot,\cdot)$ and polynomials $p(\cdot)$ and $q(\cdot)$ such that, for all x, we have

$$\begin{aligned} x \in \Pi_1 &\implies (\exists y \in \{0,1\}^n)(\forall z \in \{0,1\}^m)[A(x,y,z) = 1], \text{and} \\ x \in \Pi_2 &\implies (\forall y \in \{0,1\}^n) \Pr_{z \in \{0,1\}^m}[A(x,y,z) = 1] \leq \frac{1}{2}, \end{aligned}$$

where $n = p(|x|)$ and $m = q(|x|)$. The predicate A is called *Arthur's predicate*.

Promise AM (prAM). A promise language $\Pi = (\Pi_1, \Pi_2)$ is said to be in the promise class prAM, if there exists a polynomial time computable Boolean predicate $A(\cdot,\cdot,\cdot)$ and polynomials $p(\cdot)$ and $q(\cdot)$ such that, for all x, we have

$$\begin{aligned} x \in \Pi_1 &\implies (\forall y \in \{0,1\}^n)(\exists z \in \{0,1\}^m)[A(x,y,z) = 1], \text{and} \\ x \in \Pi_2 &\implies \Pr_{y \in \{0,1\}^n}[(\exists z \in \{0,1\}^m)A(x,y,z) = 1] \leq \frac{1}{2}, \end{aligned}$$

where $n = p(|x|)$ and $m = q(|x|)$. The predicate A is called *Arthur's predicate*.

Oracle access to promise languages. Let A be an algorithm and $\Pi = (\Pi_1, \Pi_2)$ be a promise language. When the algorithm A asks a query q, the oracle behaves as follows: if $q \in \Pi_1$, the oracle replies "yes"; if $q \in \Pi_2$, the oracle replies "no"; if q is neither in Π_1 nor in Π_2, the oracle may reply "yes" or "no". We allow the algorithm to ask queries of the third type. The requirement is that the algorithm should be able to produce the correct answer, regardless of the answers given by the oracle to the queries of the third type.

Function classes. For a promise language Π, the notation FP^{Π} refers to the class of functions that are computable by a polynomial time machine, given oracle access to Π. For a promise class $\mathcal{C}$, we denote by $\mathrm{FP}^{\mathcal{C}}$, the union of FP^{Π}, for all $\Pi \in \mathcal{C}$. Regarding $\mathrm{ZPP}^{\mathrm{NP}}$, we slightly abuse the notation and use this to mean both the standard complexity class and the function class. The function class $\mathrm{ZPP}^{\mathrm{NP}}$ contains functions computable by a zero-error probabilistic polynomial time algorithm given oracle access to NP; the algorithm either outputs a correct value of the function or "?", the latter with a small probability.

3. Main Result: Finding Collectively Irrefutable Certificates

In this section, we study the problem of finding irrefutable certificates for S_2-type matrices. As discussed in the introduction, finding a single IC in ZPP^{NP} would collapse polynomial time hierarchy (PH).

Theorem 3.1. *If there exists a* ZPP^{NP} *algorithm for the* FINDIC *problem then* PH = BPP^{NP}.

Here, we focus on finding "small" CIC's. We present an FP^{prAM} algorithm for the FINDCIC problem.

Theorem 3.2. *There exists a polynomial time algorithm which solves the following problem, given oracle access to* prAM. *The algorithm takes as input a circuit C succinctly encoding a S_2-type matrix M of size $2^n \times 2^m$ and produces either a row-side* CIC *of size m or a column-side* CIC *of size n.*

For ease of exposition, we have divided the proof into multiple small steps; in each step, the given problem is reduced (in the Turing sense) to a simpler problem. The final algorithm is obtained by composing these reductions. The various steps are grouped into two phases. The first phase reduces the given problem to a problem that we call Prefix Ratio Goodness Testing (PRGT). The second phase describes an algorithm for PRGT.

3.1. Reduction to Prefix Ratio Goodness Testing

We are given an S_2-type matrix M. By definition, M is guaranteed to have either a row-side IC or a column-side IC. Our goal is to find a small CIC. This problem reduces to the problem addressed in Lemma 3.3, given below. The lemma presents an FP^{prAM} algorithm for finding a small row-side CIC for matrices that are guaranteed to have a row-side IC. Via an easy transformation, we can obtain an analogous algorithm for finding a small column side CIC for matrices guaranteed to have a column-side IC. We run both these algorithms on the given S_2-type matrix M. Notice that one of these runs must output a CIC. The other run would output some arbitrary result, because the input matrix does not satisfy the requirements of the concerned algorithm. We check which of the two outputs is indeed a CIC and output the same. This check can be performed by making a single NP query. Thus, we get the FP^{prAM} algorithm claimed in Theorem 3.2.

Lemma 3.3. *There exists an* FP^{prAM} *algorithm that takes as input a circuit C succinctly encoding a $2^n \times 2^m$ matrix M that is guaranteed to have a row-side* IC *and outputs a row-side* CIC *of size m.*

The algorithm computes the required CIC using a standard iterative approach: in each iteration, we find a row y that beats at least half of the columns that are as yet unbeaten by the rows found in the previous iterations. Formally, we start with an empty set Y and proceed iteratively, adding a row to Y in each iteration. Consider the k^{th} iteration. Let U_k be the set of columns as yet unbeaten by any row in Y (i.e., $U_k = \{z \in \{0,1\}^m |$ no $y \in Y$ beats $z\}$). We find a row y^* such that y^* beats at least half the columns in U_k and add y^* to Y. Notice that such a y^* exists, since we are guaranteed that M has a row-side IC. Clearly, the algorithm terminates in m steps and produces a row-side CIC of size m. Of course, the main step lies in finding the required y^* in each iteration. This task is accomplished by the algorithm described in Lemma 3.4, given below. The algorithm, in fact, solves a more general problem: given *any* set of columns $X \subseteq \{0,1\}^m$, it produces a

row beating at least half of the columns in X. In each iteration, we invoke the algorithm by setting $X = U_k$. There is one minor issue that needs to be addressed: the set U_k could be exponentially large. So, we represent the set U_k in the form of a circuit C' succinctly encoding it. For this, given any column $z \in \{0,1\}^m$, C' has to test whether z is beaten by any of the rows in Y. This test involves a simulation of $C(y,z)$, for all $y \in Y$. Since Y contains at most m rows, we can succinctly encode U_k by a circuit of size polynomial in the size of C. We have proved Lemma 3.3, modulo Lemma 3.4.

Lemma 3.4. *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that takes two inputs: (i) a* $2^n \times 2^m$ *Boolean matrix* M *that is guaranteed to have a row-side* `IC`*; (ii) a set of columns* $X \subseteq \{0,1\}^m$*. It outputs a row* y^* *that beats at least half the columns in* X*. The matrix* M *and the set* X *are presented succinctly in the form of circuits.*

We build the required string y^* (of length n) in n iterations using an approach similar to self-reduction. We maintain a prefix of y^* and add one suitable bit in each iteration. However, we cannot directly employ self-reduction, since a query of the form "does there exist a row that beats at least half the columns in X" is a PP query and we cannot hope to find the answer using a prAM oracle. Nevertheless, we show how to converge on a y^* by performing self-reduction that incurs a small amount of "loss" in the "goodness" of the final y^*, in each iteration. We formalize the notion of goodness and then describe the algorithm.

Consider a $2^p \times 2^q$ Boolean matrix A and let $Q \subseteq \{0,1\}^q$ be a subset of the columns of A. For a row $y \in \{0,1\}^p$, define $\mu(y,Q)$ to be the fraction of columns in Q that y beats: $\mu(y,Q) = |\{z \in Q : y \text{ beats } z\}|/|Q|$. Let α be a string of length at most p. We say that a row $y \in \{0,1\}^p$ *extends* α, if α is a prefix of y. For $\rho \leq 1$, we say that α is ρ–good for Q, if there exists a row y extending α such that $\mu(y,Q) \geq \rho$.

The algorithm claimed in Lemma 3.4 constructs the string y^* in n iterations. Starting with the empty string, we keep building a prefix of y^*. At the end of the $(k-1)^{st}$ iteration, we have a prefix α_{k-1} of length $k-1$. In the k^{th} iteration, we extend α_{k-1} by one more bit b to get a prefix α_k of length k. To start with, we are guaranteed the existence of a row-side `IC` in M, meaning a row with goodness=1. Consider the k^{th} iteration. Suppose the prefix α_{k-1} is ρ–good with respect to X, for some ρ. Below, we describe a mechanism for finding a bit b such that the string $\alpha_{k-1}b$ is $(\rho-\epsilon)$–good. The value ϵ is a parameter to be fixed suitably later. Thus, in each iteration, we suffer a loss of ϵ and so, the accumulated loss at the end of n iterations is $n\epsilon$. Choosing ϵ suitably small, we get a string y^* having goodness at least $1/2$.

The main step lies in choosing a suitable bit b in each iteration. Consider the k^{th} iteration. At the end of $(k-1)st$ iteration, we have prefix α of length $k-1$. Write $\rho = 1-(k-1)\epsilon$. By induction, assume that α is ρ-good. Our task is to find a bit b such that αb is $(\rho-\epsilon)$-good. This is accomplished by invoking the algorithm given in Lemma 3.5, which solves the Prefix Ratio Goodness Testing problem (`PRGT`), defined below. The main observation is that at least one of $\alpha 0$ or $\alpha 1$ is ρ-good, because α is ρ-good. We run the algorithm given in Lemma 3.5 twice with $\beta = \alpha 0$ and $\beta = \alpha 1$ as inputs, respectively. By the above observation, at least one these two runs must output "yes". Let b be a bit such that the algorithm outputs "yes" on input αb. We choose b as the required bit. It is easy to see that αb is $(\rho-\epsilon)$-good; otherwise, the algorithm should have output "no" on αb.

Proceeding this way for n iterations, we end up with a string y^* which is $(1-n\epsilon)$-good. Setting $\epsilon = 1/n^2$, we see that y^* beats at least a fraction of $(1-1/n) \geq 1/2$ columns in X. We have proved Lemma 3.4, modulo Lemma 3.5.

Prefix Ratio Goodness Testing (PRGT): The instances of this promise language have four components: (i) a $2^n \times 2^m$ Boolean matrix M; (ii) a set of columns $X \subseteq \{0,1\}^m$; (iii) a prefix β of length at most n. (iv) parameters $\rho > 0$ and $\epsilon > 0$. The matrix M and the set X are represented succinctly in the form of circuits.
Positive Instances: There exists a row y extending β such that $\mu(y, X) \geq \rho$.
Negative Instances: For all rows y extending β, it is the case that $\mu(y, X) \leq \rho - \epsilon$.

Lemma 3.5. *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *that solves the* PRGT *problem. Namely, for positive instances, the output is "yes"; for negative instances, the output is "no"; for other instances, the output can be arbitrary. The running time of the algorithm has a polynomial dependence on* $1/\epsilon$.

3.2. Prefix Ratio Goodness Testing: Proof of Lemma 3.5

In this section, we prove Lemma 3.5. One of the hurdles in trying to construct the desired algorithm is that the gap between the two cases we need to distinguish is small. So, as a first step, we amplify the gap using standard techniques.

The amplification process involves a parameter t, which we will fix suitably. Construct a matrix $\overline{M}$ from M as follows. Each row in $\overline{M}$ corresponds to a row in M and each column $\overline{z}$ in $\overline{M}$ corresponds to a sequence $\langle z_1, z_2, \ldots, z_t \rangle$ of t columns from M. Thus, the matrix $\overline{M}$ is of size $2^n \times 2^{\overline{m}}$, where $\overline{m} = mt$. Consider a row $y \in \{0,1\}^n$ and a column $\overline{z} = \langle z_1, z_2, \ldots, z_t \rangle$, where each z_i is a column in M. Set the entry $\overline{M}[y, \overline{z}] = 1$, if y beats at least $(\rho - \frac{\epsilon}{2})$ fraction of the z_i's (with respect to M); otherwise, set it to 0. Analogously, denote by $\overline{X}$ the t-wise cartesian product of X with itself, i.e., $\overline{X} = \{\langle z_1, z_2, \ldots, z_t \rangle \ : \ z_i \in X\}$. We fix $t = 16m/\epsilon^2$. An application of Chernoff bounds yields the following claim.

Lemma 3.6. *For any* $y \in \{0,1\}^n$, *we have the following.*

- *If* $\mu(y, X) \geq \rho$ *in* M *then* $\mu(y, \overline{X}) \geq 1/2$ *in* $\overline{M}$.
- *If* $\mu(y, X) \leq \rho - \epsilon$ *in* M *then* $\mu(y, \overline{X}) \leq 1/\overline{m}^4$ *in* $\overline{M}$.

Given the above amplification, the problem considered in Lemma 3.5 reduces to the problem addressed in Lemma 3.7. Formally, the algorithm claimed in Lemma 3.5 works as follows. Given a circuit C succinctly encoding the matrix M, a circuit C_X succintly encoding a set of columns X, prefix β and parameters ρ and ϵ, we consider the matrix $\overline{M}$ and the set $\overline{X}$, as described above. Notice that we can construct in polynomial time a circuit $\overline{C}$ succinctly encoding $\overline{M}$ such that $|\overline{C}|$ is polynomial in $|C|$ and $1/\epsilon$. Similarly, we can construct a polynomial size circuit $\overline{C}_X$ succinctly encoding the set $\overline{X}$. Then, we invoke the algorithm given in Lemma 3.7 with $\overline{C}$, $\overline{C}_X$ and β as inputs. We output "yes", if the algorithm outputs "yes" and output "no", otherwise. This completes the proof of Lemma 3.5, modulo Lemma 3.7.

Lemma 3.7. *There exists an* $\mathrm{FP}^{\mathrm{prAM}}$ *algorithm that takes three inputs: (i) a* $2^n \times 2^m$ *Boolean matrix* M; *(ii) a set of columns* $X \subseteq \{0,1\}^m$. *(iii) a prefix* β *of length at most* n. *The matrix* M *and the set* X *are presented succinctly in the form of circuits. The algorithm has the following property:*

- *Case (a) : If there exists a row* y *extending* β *such that* $\mu(y, X) \geq 1/2$, *then it outputs "yes".*
- *Case (b) : If all rows* y *extending* β *are such that* $\mu(y, X) \leq 1/m^4$, *then it outputs "no".*

If neither of the above conditions is true, then the output of the algorithm is arbitrary.

There are two main stages in the algorithm. In the first stage we get an estimate on the size of X. And in the second stage, we use the above estimate to distinguish between the cases (a) and (b) in the lemma. Both the stages make queries to a prAM language given as oracle. A lemma, due to Sipser [20], is useful in establishing that the concerned language indeed lies in the class prAM. The following notation is needed for describing the lemma.

Let $\mathcal{H}$ be a family of functions mapping $\{0,1\}^m$ to $\{0,1\}^k$. Recall that $\mathcal{H}$ is said to be 2-universal, if for any $z, z' \in \{0,1\}^m$, with $z \neq z'$, and $x, x' \in \{0,1\}^k$, $\Pr_{h\in\mathcal{H}}[h(z) = x \text{ and } h(z') = x'] = 1/2^{2k}$. It is well known that such a family can easily be constructed. For instance, the set of all $m \times k$ Boolean matrices yield such a family; a matrix B represents the function h given by $h(z) = zB$ (modulo 2).

For a function $h \in \mathcal{H}$ and a string $z \in \{0,1\}^m$, we say that z has a *collision* under h, if there exists a $z' \in \{0,1\}^m$ such that $z \neq z'$ and $h(z) = h(z')$. For a set of hash functions $H \subseteq \mathcal{H}$, we say that z has a *collision* under H, if for all $h \in H$, z has a collision under h. A set $S \subseteq \{0,1\}^m$ is said to have a collision under H, if there exists a $z \in S$ such that z has a collision under H.

Lemma 3.8 ([20])**.** *Let $S \subseteq \{0,1\}^m$ and $k \leq m$. Let $\mathcal{H}$ be a 2-universal family of hash functions from $\{0,1\}^m$ to $\{0,1\}^k$. Uniformly and independently pick a set of hash functions $h_1, h_2, \ldots, h_k$ from $\mathcal{H}$ and let $H = \{h_1, h_2, \ldots, h_k\}$. Then,*

- *If $|S| > k2^k$, then $\Pr_H[S \text{ has a collision under } H] = 1$.*
- *If $|S| \leq 2^{k-1}$, then $\Pr_H[S \text{ has a collision under } H] \leq 1/2$.*

We define a promise language called *set largeness testing* (SLT) and then use Lemma 3.8 to show that it lies in the class prAM.

Set Largeness Testing (SLT): The instances in this language consist of a set $X \subseteq \{0,1\}^m$, presented succinctly in the form of a circuit, and an integer $k \leq m$.
Positive instances: $|X| > k2^k$.
Negative instances: $|X| \leq 2^{k-1}$.

Lemma 3.9. *The promise language* SLT *belongs to the class* prAM.

Proof. Let $\mathcal{H}$ be a 2-universal family of hash functions from $\{0,1\}^m$ to $\{0,1\}^k$. The proof is based on the observation that for a given set $H \subseteq \mathcal{H}$, testing whether X has a collision under H is an NP predicate.

The AM protocol proceeds as follows. Arthur picks a set of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ uniformly and independently at random from $\mathcal{H}$. Merlin must exhibit an element $z \in X$ and prove that z has a collision under H. Arthur accepts, if Merlin proves that such a collision exists; otherwise, Arthur rejects. □

The following lemma provides an algorithm for estimating the size of a set, given SLT as oracle.

Lemma 3.10. *There exists an $\mathrm{FP}^{\mathrm{prAM}}$ that takes a set $X \subseteq \{0,1\}^m$, presented succinctly in the form of a circuit, and outputs an estimate U such that $\frac{|U|}{4m} \leq |X| \leq |U|$.*

Proof. The algorithm takes the promise language SLT as the oracle. We iteratively consider every integer k in the range 1 through m and ask the query (X, k) to the oracle. Let k_e be the first time, we get a "no" answer from the oracle. Compute $|U| = m2^{k_e}$. We shall argue that U satisfies the stated bounds.

Let k_0 be the largest integer such that $|X| > k_0 2^{k_0}$ and let k_1 be the smallest integer such that $|X| \leq 2^{k_1-1}$. Notice that $k_0 + 1 \leq k_e \leq k_1$. By the property of k_0, k_e satisfies $|X| \leq k_e 2^{k_e} \leq m2^{k_e}$. By the property of k_1, we have that $2^{k_1-2} < |X| \leq 2^{k_1-1}$. It follows that $2^{k_e} \leq 2^{k_1} < 4|X|$. The claimed bounds on $|U|$ follow from the above inequalities. □

Returning to Lemma 3.7, the first stage of the algorithm (finding an estimate on $|X|$) can now be performed using Lemma 3.10. We turn to the second stage that involves distinguishing between the two cases in Lemma 3.7. For this, we will make use of the following promise language as an oracle.

Prefix Cardinality Goodness Testing (PCGT): The instances of this language consist of four components: (i) a $2^n \times 2^m$ Boolean matrix M; (ii) a set $X \subseteq \{0,1\}^m$; (iii) a prefix β of length at most n; (iv) a number k. The matrix M and the set X are presented succinctly in the form of circuits.

Positive instances: There exists a row y extending β such that y beats at least $k2^k$ columns in X.

Negative instances: For all rows y extending β, y beats at most 2^{k-1} columns in X.

Lemma 3.11. *The promise language* PCGT *belongs to the class* prAM.

Proof. The proof is similar to that of Lemma 3.9 and makes use of Lemma 3.8. We present an MAM protocol. It is well known that such a protocol can be converted to an AM protocol [3].

Merlin claims that a given instance is of the positive type. To prove this, he provides a row y extending β. Let $Z \subseteq X$ be the set of columns from X that are beaten by y. Arthur needs to distinguish between the cases of $|Z| > k2^k$ and $|Z| \leq 2^{k-1}$. This situation is the same as that of Lemma 3.9. By repeating the argument from there, we get an MAM protocol. □

Proof of Lemma 3.7: Our algorithm will make use of both SLT and PCGT as oracles. Let us rephrase the two cases that we wish to distinguish:

- Case (a): There exists a row y extending β such that y beats at least $|X|/2$ columns from X.
- Case (b) : For any row y extending β, y beats at most $|X|/m^4$ columns from X.

We first run the algorithm claimed in Lemma 3.10 to get an estimate U such that $|U|/4m \leq |X| \leq |U|$. Our next goal is to reduce the task of distinguishing the above two cases to a PCGT query. Consider any row y. Let Z be the number of columns from X that y beats. We wish to choose a number k satisfying two conditions: (i) if $Z \geq |X|/2$ then $Z > k2^k$; (ii) if $Z \leq |X|/m^4$ then $Z \leq 2^{k-1}$. A simple calculation reveals that it suffices for k to satisfy the following inequalities in terms of U:

$$\frac{2U}{m^4} \leq 2^k \leq \frac{U}{8m^2}.$$

Clearly, we can choose $k = \lfloor \log \frac{U}{8m^2} \rfloor$. Then, we call the PCGT oracle with the parameters M, X, β and k. We output "yes", if the oracle says "yes"; and output "no", if the oracle says "no". □

4. Applications of the Main Result

In this section, we apply Theorem 3.2 in two different settings and derive some corollaries. The first deals with upperbounds on the power of S_2^p. The second is about the consequences of NP having polynomial size circuits.

4.1. Upperbounds for S_2^p

Theorem 4.1. $S_2^p \subseteq P^{prAM}$.

Proof. The claim follows directly from Theorem 3.2. Let L be a language in S_2^p. Let x be the input string. Consider the S_2-type matrix M corresponding to x. As discussed in Section 2, we can obtain a circuit C succinctly encoding the matrix M in time polynomial in $|x|$. Invoking the algorithm given in Theorem 3.2 on C, we get either a row-side `CIC` or a column-side `CIC`. Notice that in the former case $x \in L$ and in the latter case $x \notin L$. □

Having proven the above theorem, it is natural to ask how large the class P^{prAM} is. By definition, AM is contained in BPP^{NP} and so, P^{AM} is also contained in the same class. We observe the above claim extends to the case where the oracle is a prAM oracle, instead of an AM oracle.

Theorem 4.2. $P^{prAM} \subseteq BPP^{NP}$.

Cai [5] showed that S_2^p is contained in ZPP^{NP}, whereas our result puts S_2^p in the class P^{prAM}. The containment relationships between ZPP^{NP} and P^{prAM} are unknown. In this context, we observe that an alternative proof of Cai's result can be derived using our techniques. This cannot be achieved by simply combining Theorem 4.1 and 4.2; this would only yield $S_2^p \subseteq BPP^{NP}$. We obtain the alternative proof by directly appealing to Theorem 3.2.

Theorem 4.3 ([5])**.** $S_2^p \subseteq ZPP^{NP}$.

4.2. Consequences of NP having small circuits

A body of prior work has dealt with the implication of the assumption that NP has polynomial size circuits. Our main theorem yields some new results in this context, which are described in this section.

Suppose NP is contained in P/poly. Karp and Lipton [13] showed that, under this assumption, the polynomial time hierarchy (PH) collapses to $\Sigma_2^p \cap \Pi_2^p$, i.e., $PH = \Sigma_2^p \cap \Pi_2^p$. Köbler and Watanabe [14] improved the collapse to ZPP^{NP}. Sengupta (see [5]) observed that the collapse can be brought down to S_2^p. This has been further improved via a collapse to O_2^p, the oblivious version of S_2^p [7]. It has been an interesting open problem to obtain a collapse to the class P^{NP}. Here, we show a collapse to P^{prMA}.

Theorem 4.4. *If* $NP \subseteq P/poly$*, then* $PH = P^{prMA}$.

Proof. By Sengupta's observation [5], the assumption implies that $PH = S_2^p$. Combining this with Theorem 4.1, we get $PH = P^{prAM}$. Arvind et al. [2] showed that if $NP \subseteq P/poly$ then $AM = MA$. We observe that this result carries over to the promise versions, namely the same assumption implies $prAM = prMA$. The claim follows. □

Though the above theorem yields a new consequence, we note that it is not clear whether this is an improvement over the previously best known collapse. It is known that $MA \subseteq ZPP^{NP}$ [11, 1] and $MA \subseteq S_2^p$ [17]. Extending the former claim, we can show that $P^{prMA} \subseteq ZPP^{NP}$. However, we do not know how to accomplish the same for the second claim. Namely, it remains open whether P^{prMA} is contained in S_2^p.

Under the assumption NP has polynomial size circuits, Bshouty et al. [4] studied the problem of learning a correct circuit for SAT and designed a ZPP^{NP} algorithm. Using Theorem 3.2, we derive the following claim which improves the above result, as we can show that $FP^{prMA} \subseteq ZPP^{NP}$.

Theorem 4.5. *If* NP $\subseteq$ P/poly*, then there exists an* $\mathrm{FP}^{\mathrm{prMA}}$ *algorithm that outputs a correct polynomial size circuit for* SAT *at a given input length.*

Acknowledgments: We thank the anonymous referees for their useful comments.

References

[1] V. Arvind and J. Köbler. On pseudorandomness and resource-bounded measure. In *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science*, 1997.

[2] V. Arvind, J. Köbler, U. Schöning, and R. Schuler. If NP has polynomial-size circuits, then MA=AM. *Theoretical Computer Science*, 137(2):279–282, 1995.

[3] L. Babai and S. Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *Journal of Computer and System Sciences*, 36(2):254–276, 1988.

[4] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, 1996.

[5] J. Cai. $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. *Journal of Computer and System Sciences*, 73(1), 2007.

[6] R. Canetti. More on BPP and the polynomial-time hierarchy. *Information Processing Letters*, 57(5):237–241, 1996.

[7] V. Chakaravarthy and S. Roy. Oblivious symmetric alternation. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science*, 2006.

[8] D. Du and K. Ko. *Computational Complexity*. John Wiley and sons, 2000.

[9] L. Fortnow, R. Impagliazzo, V. Kabanets, and C. Umans. On the complexity of succinct zero-sum games. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, 2005.

[10] O. Goldreich and A. Wigderson. Improved derandomization of BPP using a hitting set generator. In *RANDOM-APPROX*, 1999.

[11] O. Goldreich and D. Zuckerman. Another proof that BPP $\subseteq$ PH (and more). Technical Report TR97–045, ECCC, 1997.

[12] M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43, 1986.

[13] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, 1980.

[14] J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM Journal on Computing*, 28(1):311–324, 1998.

[15] P. Miltersen and N. Vinodchandran. Derandomizing Arthur-Merlin games using hitting sets. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, 1999.

[16] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[17] A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Computational Complexity*, 7(2):152–162, 1998.

[18] R. Santhanam. Circuit lower bounds for Merlin-Arthur classes. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, 2007.

[19] R. Shaltiel and C. Umans. Pseudorandomness for approximate counting and sampling. *Computational Complexity*, 15(4), 2007.

[20] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, 1983.

[21] L. Stockmeyer. The complexity of approximate counting. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, 1983.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 169-180
www.stacs-conf.org

QUANTIFYING HOMOLOGY CLASSES

CHAO CHEN [1] AND DANIEL FREEDMAN [1]

[1] Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180, U.S.A.
E-mail address, {C. Chen, D. Freedman}: {chenc3,freedman}@cs.rpi.edu

Abstract. We develop a method for measuring homology classes. This involves three problems. First, we define the size of a homology class, using ideas from relative homology. Second, we define an optimal basis of a homology group to be the basis whose elements' size have the minimal sum. We provide a greedy algorithm to compute the optimal basis and measure classes in it. The algorithm runs in $O(\beta^4 n^3 \log^2 n)$ time, where n is the size of the simplicial complex and β is the Betti number of the homology group. Third, we discuss different ways of localizing homology classes and prove some hardness results.

1. Introduction

The problem of computing the topological features of a space has recently drawn much attention from researchers in various fields, such as high-dimensional data analysis [3, 15], graphics [13, 5], networks [10] and computational biology [1, 8]. Topological features are often preferable to purely geometric features, as they are more qualitative and global, and tend to be more robust. If the goal is to characterize a space, therefore, features which incorporate topology seem to be good candidates.

Once we are able to compute topological features, a natural problem is to rank the features according to their importance. The significance of this problem can be justified from two perspectives. First, unavoidable errors are introduced in data acquisition, in the form of traditional signal noise, and finite sampling of continuous spaces. These errors may lead to the presence of many small topological features that are not "real", but are simply artifacts of noise or of sampling [19]. Second, many problems are naturally hierarchical. This hierarchy – which is a kind of multiscale or multi-resolution decomposition – implies that we want to capture the large scale features first. See Figure 1(a) and 1(b) for examples.

The topological features we use are homology groups over $\mathbb{Z}_2$, due to their ease of computation. (Thus, throughout this paper, all the additions are mod 2 additions.) We would then like to quantify or measure homology classes, as well as collections of classes. Specifically, there are three problems we would like to solve:

(1) **Measuring the size of a homology class:** We need a way to quantify the size of a given homology class, and this size measure should agree with intuition. For example, in Figure 1(a), the measure should be able to distinguish the one large class (of the 1-dimensional homology group) from the two smaller classes. Furthermore,

1998 ACM Subject Classification: F.2.2, G.2.1.

Key words and phrases: Computational Topology, Computational Geometry, Homology, Persistent Homology, Localization, Optimization.

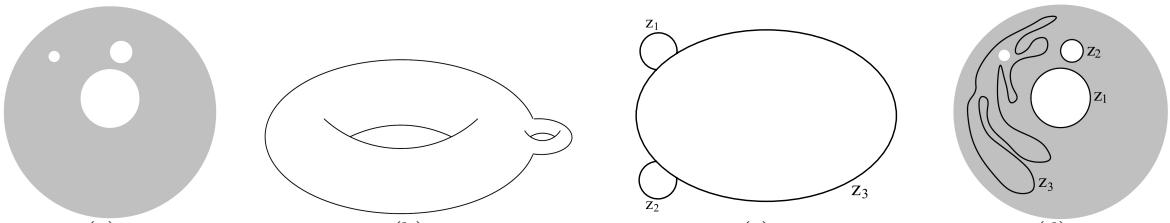


Figure 1: (a,b) A disk with three holes and a 2-handled torus are really more like an annulus and a 1-handled torus, respectively, because the large features are more important. (c) A topological space formed from three circles. (d) In a disk with three holes, cycles z_1 and z_2 are well-localized; z_3 is not.

the measure should be easy to compute, and applicable to homology groups of any dimension.

(2) **Choosing a basis for a homology group:** We would like to choose a "good" set of homology classes to be the generators for the homology group (of a fixed dimension). Suppose that β is the dimension of this group, and that we are using $\mathbb{Z}_2$ coefficients; then there are $2^\beta - 1$ nontrivial homology classes in total. For a basis, we need to choose a subset of β of these classes, subject to the constraint that these β generate the group. The criterion of goodness for a basis is based on an overall size measure for the basis, which relies in turn on the size measure for its constituent classes. For instance, in Figure 1(c), we must choose three from the seven nontrivial 1-dimensional homology classes: $\{[z_1], [z_2], [z_3], [z_1]+[z_2], [z_1]+[z_3], [z_2]+[z_3], [z_1]+[z_2]+[z_3]\}$. In this case, the intuitive choice is $\{[z_1], [z_2], [z_3]\}$, as this choice reflects the fact that there is really only one large cycle.

(3) **Localization:** We need the smallest cycle to represent a homology class, given a natural criterion of the size of a cycle. The criterion should be deliberately chosen so that the corresponding smallest cycle is both mathematically natural and intuitive. Such a cycle is a "well-localized" representative of its class. For example, in Figure 1(d), the cycles z_1 and z_2 are well-localized representatives of their respective homology classes; whereas z_3 is not.

Furthermore, we make two additional requirements on the solution of aforementioned problems. First, the solution ought to be computable for topological spaces of arbitrary dimension. Second the solution should not require that the topological space be embedded, for example in a Euclidean space; and if the space is embedded, the solution should not make use of the embedding. These requirements are natural from the theoretical point of view, but may also be justified based on real applications. In machine learning, it is often assumed that the data lives on a manifold whose dimension is much smaller than the dimension of the embedding space. In the study of shape, it is common to enrich the shape with other quantities, such as curvature, or color and other physical quantities. This leads to high dimensional manifolds (e.g, 5-7 dimensions) embedded in high dimensional ambient spaces [4].

Although there are existing techniques for approaching the problems we have laid out, to our knowledge, there are no definitions and algorithms satisfying the two requirements. Ordinary persistence [12, 20, 6] provides a measure of size, but only for those *inessential*

classes, i.e. classes which ultimately die. More recent work [7] attempts to remedy this situation, but not in an intuitive way. Zomorodian and Carlsson [21] use advanced algebraic topological machinery to solve the basis computation and localization problems. However, both the quality of the result and the complexity depend strongly on the choice of the given cover; there is, as yet, no suggestion of a canonical cover. Other works like [14, 19, 11] are restricted to low dimension.

Contributions. In this paper, we solve these problems. Our contributions include:

- Definitions of the size of homology classes and the optimal homology basis.
- A provably correct greedy algorithm to compute the optimal homology basis and measure its classes. This algorithm uses the persistent homology.
- An improvement of the straightforward algorithm using finite field linear algebra.
- Hardness results concerning the localization of homology classes.

2. Defining the Problem

In this section, we provide a technique for ranking homology classes according to their importance. Specifically, we solve the first two problems mentioned in Section 1 by formally defining (1) a meaningful size measure for homology classes that is computable in arbitrary dimension; and (2) an optimal homology basis which distinguishes large classes from small ones effectively.

Since we restrict our work to homology groups over $\mathbb{Z}_2$, when we talk about a d-dimensional chain, c, we refer to either a collection of d-simplices, or a n_d-dimensional vector over $\mathbb{Z}_2$ field, whose non-zero entries corresponds to the included d-simplices. n_d is the number of d-dimensional simplces in the given complex, K. The relevant background in homology and relative homology can be found in [16].

The Discrete Geodesic Distance. In order to measure the size of homology classes, we need a notion of distance. As we will deal with a simplicial complex K, it is most natural to introduce a discrete metric, and corresponding distance functions. We define the *discrete geodesic distance* from a vertex $p \in \mathrm{vert}(K)$, $f_p : \mathrm{vert}(K) \to \mathbb{Z}$, as follows. For any vertex $q \in \mathrm{vert}(K)$, $f_p(q) = \mathrm{dist}(p, q)$ is the length of the shortest path connecting p and q, in the 1-skeleton of K; it is assumed that each edge length is one, though this can easily be changed. We may then extend this distance function from vertices to higher dimensional simplices naturally. For any simplex $\sigma \in K$, $f_p(\sigma)$ is the maximal function value of the vertices of σ, $f_p(\sigma) = \max_{q \in \mathrm{vert}(\sigma)} f_p(q)$. Finally, we define a *discrete geodesic ball* B_p^r, $p \in \mathrm{vert}(K)$, $r \geq 0$, as the subset of K, $B_p^r = \{\sigma \in K \mid f_p(\sigma) \leq r\}$. It is straightforward to show that these subsets are in fact *subcomplexes*, namely, subsets that are still simplicial complexes.

2.1. Measuring the Size of a Homology Class

We start this section by introducing notions from relative homology. Given a simplicial complex K and a subcomplex $L \subseteq K$, we may wish to study the structure of K by ignoring all the chains in L. We study the *group of relative chain* as a quotient group, $\mathsf{C}_d(K, L) = \mathsf{C}_d(K)/\mathsf{C}_d(L)$, whose elements are *relative chains*. Analogous to the way we define the group of cycles $\mathsf{Z}_d(K)$, the group of boundaries $\mathsf{B}_d(K)$ and the homology group $\mathsf{H}_d(K)$ in $\mathsf{C}_d(K)$, we

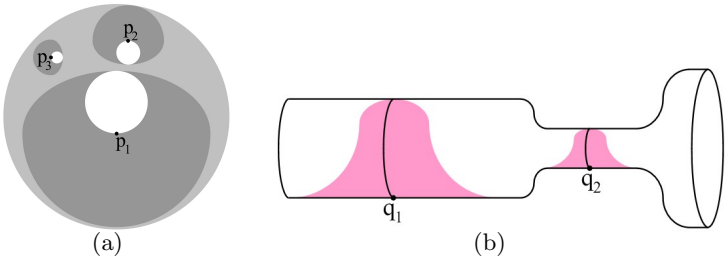


Figure 2: (a) On a disk with three holes, the three shaded regions are the three smallest geodesic balls measuring the three corresponding classes. (b) On a tube, the smallest geodesic ball is centered at q_2, not q_1.

define the *group of relative cycles*, the *group of relative boundaries* and the *relative homology group* in $\mathsf{C}_d(K, L)$, denoted as $\mathsf{Z}_d(K, L)$, $\mathsf{B}_d(K, L)$ and $\mathsf{H}_d(K, L)$, respectively. We denote $\phi_L : \mathsf{C}_d(K) \to \mathsf{C}_d(K, L)$ as the homomorphism mapping d-chains to their corresponding relative chains, $\phi_L^* : \mathsf{H}_d(K) \to \mathsf{H}_d(K, L)$ as the induced homomorphism mapping homology classes of K to their corresponding relative homology classes.

Using these notions, we define the size of a homology class as follows. Given a simplicial complex K, assume we are given a collection of subcomplexes $\mathcal{L} = \{L \subseteq K\}$. Furthermore, each of these subcomplexes is endowed with a size. In this case, we define the size of a homology class h as the size of the smallest L carrying h. Here we say a subcomplex L *carries* h if h has a trivial image in the relative homology group $\mathsf{H}_d(K, L)$, formally, $\phi_L^*(h) = \mathsf{B}_d(K, L)$. Intuitively, this means that h disappears if we delete L from K, by contracting it into a point and modding it out.

Definition 2.1. The size of a class h, $S(h)$, is the size of the smallest measurable subcomplex carrying h, formally, $S(h) = \min_{L \in \mathcal{L}} \text{size}(L)$ such that $\phi_L^*(h) = \mathsf{B}_d(K, L)$.

We say a subcomplex L *carries* a chain c if L contains all the simplices of the chain, formally, $c \subseteq L$. Using standard facts from algebraic topology, it is straightforward to see that L carries h if and only if it carries a cycle of h. This gives us more intuition behind the measure definition.

In this paper, we take $\mathcal{L}$ to be the set of discrete geodesic balls, $\mathcal{L} = \{B_p^r \mid p \in \text{vert}(K), r \geq 0\}$.[1] The size of a geodesic ball is naturally its radius r. The smallest geodesic ball carrying h is denoted as $B_{min}(h)$ for convenience, whose radius is $S(h)$. In Figure 2(a), the three geodesic balls centered at p_1, p_2 and p_3 are the smallest geodesic balls carrying nontrivial homology classes $[z_1]$, $[z_2]$ and $[z_3]$, respectively. Their radii are the size of the three classes. In Figure 2(b), the smallest geodesic ball carrying a nontrivial homology class is the pink one centered at q_2, not the one centered at q_1. Note that these geodesic balls may not look like Euclidean balls in the embedding space.

2.2. The Optimal Homology Basis

For the d-dimensional $\mathbb{Z}_2$ homology group whose dimension (Betti number) is β_d, there are $2^{\beta_d} - 1$ nontrivial homology classes. However, we only need β_d of them to form a basis.

[1]The idea of growing geodesic discs has been used in [19]. However, this work depends on low dimensional geometric reasoning, and hence is restricted to 1-dimensional homology classes in 2-manifold.

The basis should be chosen wisely so that we can easily distinguish important homology classes from noise. See Figure 1(c) for an example. There are $2^3-1=7$ nontrivial homology classes; we need three of them to form a basis. We would prefer to choose $\{[z_1],[z_2],[z_3]\}$ as a basis, rather than $\{[z_1]+[z_2]+[z_3],[z_2]+[z_3],[z_3]\}$. The former indicates that there is one big cycle in the topological space, whereas the latter gives the impression of three large classes.

In keeping with this intuition, the *optimal homology basis* is defined as follows.

Definition 2.2. The optimal homology basis is the basis for the homology group whose elements' size have the minimal sum, formally,

$$\mathcal{H}_d = \operatorname*{argmin}_{\{h_1,\ldots,h_{\beta_d}\}} \sum_{i=1}^{\beta_d} S(h_i), s.t. \dim(\{h_1,\ldots,h_{\beta_d}\}) = \beta_d.$$

This definition guarantees that large homology classes appear as few times as possible in the optimal homology basis. In Figure 1(c), the optimal basis will be $\{[z_1],[z_2],[z_3]\}$, which has only one large class.

For each class in the basis, we need a cycle representing it. As we has shown, $B_{min}(h)$, the smallest geodesic ball carrying h, carries at least one cycle of h. We localize each class in the optimal basis by its *localized-cycles*, which are cycles of h carried by $B_{min}(h)$. This is a fair choice because it is consistent to the size measure of h and it is computable in polynomial time. See Section 5 for further discussions.

3. The Algorithm

In this section, we introduce an algorithm to compute the optimal homology basis as defined in Definition 2.2. For each class in the basis, we measure its size, and represent it with one of its localized-cycles. We first introduce an algorithm to compute the smallest homology class, namely, Measure-Smallest(K). Based on this procedure, we provide the algorithm Measure-All(K), which computes the optimal homology basis. The algorithm takes $O(\beta_d^4 n^4)$ time, where β_d is the Betti number for d-dimensional homology classes and n is the cardinality of the input simplicial complex K.

Persistent Homology. Our algorithm uses the persistent homology algorithm. In persistent homology, we filter a topological space with a scalar function, and capture the birth and death times of homology classes of the sublevel set during the filtration course. Classes with longer persistences are considered important ones. Classes with infinite persistences are called *essential homology classes* and corresponds to the intrinsic homology classes of the given topological space. Please refer to [12, 20, 6] for theory and algorithms of persistent homology.

3.1. Computing the Smallest Homology Class

The procedure Measure-Smallest(K) measures and localizes, h_{min}, the smallest nontrivial homology class, namely, the one with the smallest size. The output of this procedure will be a pair (S_{min}, z_{min}), namely, the size and a localized-cycle of h_{min}. According to the definitions, this pair is determined by the smallest geodesic ball carrying h_{min}, namely,

$B_{min}(h_{min})$. We first present the algorithm to compute this ball. Second, we explain how to compute the pair (S_{min}, z_{min}) from the computed ball.

Procedure Bmin(K): Computing $B_{min}(h_{min})$. It is straightforward to see that the ball $B_{min}(h_{min})$ is also the smallest geodesic ball carrying any nontrivial homology class of K. It can be computed by computing $B_p^{r(p)}$ for all vertices p, where $B_p^{r(p)}$ is the smallest geodesic ball centered at p which carries any nontrivial homology class. When all the $B_p^{r(p)}$'s are computed, we compare their radii, $r(p)$'s, and pick the smallest ball as $B_{min}(h_{min})$.

For each vertex p, we compute $B_p^{r(p)}$ by applying the persistent homology algorithm to K with the discrete geodesic distance from p, f_p, as the filter function. Note that a geodesic ball B_p^r is the sublevel set $f_p^{-1}(-\infty, r] \subseteq K$. Nontrivial homology classes of K are essential homology classes in the persistent homology algorithm. (In the rest of this paper, we may use "essential homology classes" and "nontrivial homology classes of K" interchangable.) Therefore, the birth time of the first essential homology class is $r(p)$, and the subcomplex $f_p^{-1}(-\infty, r(p)]$ is $B_p^{r(p)}$.

Computing (S_{min}, z_{min}). We compute the pair from the computed ball $B_{min}(h_{min})$. For simplicity, we denote p_{min} and r_{min} as the center and radius of the ball. According to the definition, r_{min} is exactly the size of h_{min}, S_{min}. Any nonbounding cycle (a cycle that is not a boundary) carried by $B_{min}(h_{min})$ is a localized-cycle of h_{min}.[2] We first computes a basis for all cycles carried by $B_{min}(h_{min})$, using a reduction algorithm. Next, elements in this basis are checked one by one until we find one which is nounbounding in K. This checking uses the algorithm of Wiedemann[18] for rank computation of sparse matrices over $\mathbb{Z}_2$ field.

3.2. Computing the Optimal Homology Basis

In this section, we present the algorithm for computing the optimal homology basis defined in Definition 2.2, namely, $\mathcal{H}_d$. We first show that the optimal homology basis can be computed in a greedy manner. Second, we introduce an efficient greedy algorithm.

3.2.1. *Computing $\mathcal{H}_d$ in a Greedy Manner.* Recall that the optimal homology basis is the basis for the homology group whose elements' size have the minimal sum. We use matroid theory [9] to show that we can compute the optimal homology basis with a greedy method. Let H be the set of nontrivial d-dimensional homology classes (i.e. the homology group minus the trivial class). Let L be the family of sets of linearly independent nontrivial homology classes. Then we have the following theorem, whose proof is omitted due to space limitations. The same result has been mentioned in [14].

Theorem 3.1. *The pair (H, L) is a matroid when $\beta_d > 0$.*

We construct a weighted matroid by assigning each nontrivial homology class its size as the weight. This weight function is strictly positive because a nontrivial homology class can not be carried by a geodesic ball with radius zero. According to matroid theory, we can compute the optimal homology basis with a naive greedy method: check the smallest nontrivial homology classes one by one, until β_d linearly independent ones are collected.

[2]This is true assuming that $B_{min}(h_{min})$ carries one and only one nontrivial class, i.e. h_{min} itself. However, it is straightforward to relax this assumption.

The collected β_d classes $\{h_{i_1}, h_{i_2}, ..., h_{i_{\beta_d}}\}$ form the optimal homology basis $\mathcal{H}_d$. (Note that the h's are ordered by size, i.e. $S(h_{i_k}) \leq S(h_{i_{k+1}})$.) However, this method is exponential in β_d. We need a better solution.

3.2.2. *Computing $\mathcal{H}_d$ with a Sealing Technique.* In this section, we introduce a polynomial greedy algorithm for computing $\mathcal{H}_d$. Instead of computing the smallest classes one by one, our algorithm uses a sealing technique and takes time polynomial in β_d. Intuitively, when the smallest l classes in $\mathcal{H}_d$ are picked, we make them trivial by adding new simplices to the given complex. In the augmented complex, any linear combinations of these picked classes becomes trivial, and the smallest nontrivial class is the $(l+1)$'th one in $\mathcal{H}_d$.

The algorithm starts by measuring and localizing the smallest homology class of the given simplicial complex K (using the procedure Measure-Smallest(K) introduced in Section 3.1), which is also the first class we choose for $\mathcal{H}_d$. We make this class trivial by sealing one of its cycles – i.e. the localized-cycle we computed – with new simplices. Next, we measure and localize the smallest homology class of the augmented simplicial complex K'. This class is the second smallest homology class in $\mathcal{H}_d$. We make this class trivial again and proceed for the third smallest class in $\mathcal{H}_d$. This process is repeated for β_d rounds, yielding $\mathcal{H}_d$.

We make a homology class trivial by sealing the class's localized-cycle, which we have computed. To seal this cycle z, we add (a) a new vertex v; (b) a $(d+1)$-simplex for each d-simplex of z, with vertex set equal to the vertex set of the d-simplex together with v; (c) all of the faces of these new simplices. In Figure 3(a) and 3(b), a 1-cycle with four edges, z_1, is sealed up with one new vertex, four new triangles and four new edges.

It is essential to make sure the new simplices does not influence our measurement. We assign the new vertices $+\infty$ geodesic distance from any vertex in the original complex K. Furthermore, in the procedure Measure-Smallest(K'), we will not consider any geodesic ball centered at these new vertices. In other words, the geodesic distance from these new vertices will never be used as a filter function. Whenever we run the persistent homology algorithm, all of the new simplices have $+\infty$ filter function values, formally, $f_p(\sigma) = +\infty$ for all $p \in \text{vert}(K)$ and $\sigma \in K' \backslash K$.

The algorithm is illustrated in Figure 3(a) and 3(b). The 4-edge cycle, z_1, and the 8-edge cycle, z_2, are the localized-cycles of the smallest and the second smallest homology classes ($S([z_1]) = 2$,$S([z_2]) = 4$). The nonbounding cycle $z_3 = z_1 + z_2$ corresponds to the largest nontrivial homology class $[z_3] = [z_1] + [z_2]$ ($S([z_3]) = 5$). After the first round, we choose $[z_1]$ as the smallest class in $\mathcal{H}_1$. Next, we destroy $[z_1]$ by sealing z_1, which yields the augmented complex K'. This time, we choose $[z_2]$, giving $\mathcal{H}_1 = \{[z_1], [z_2]\}$.

Correctness. We prove in Theorem 3.3 the correctness of our greedy method. We begin by proving a lemma that destroying the smallest nontrivial class will neither destroy any other classes nor create any new classes. Please note that this is not a trivial result. The lemma does not hold if we seal an arbitrary class instead of the smallest one. See Figure 3(c) and 3(d) for examples. Our proof is based on the assumption that the smallest nontrivial class h_{min} is the only one carried by $B_{min}(h_{min})$.

Lemma 3.2. *Given a simplicial complex K, if we seal its smallest homology class $h_{min}(K)$, any other nontrivial homology class of K, h, is still nontrivial in the augmented simplicial complex K'. In other words, any cycle of h is still nonbounding in K'.*

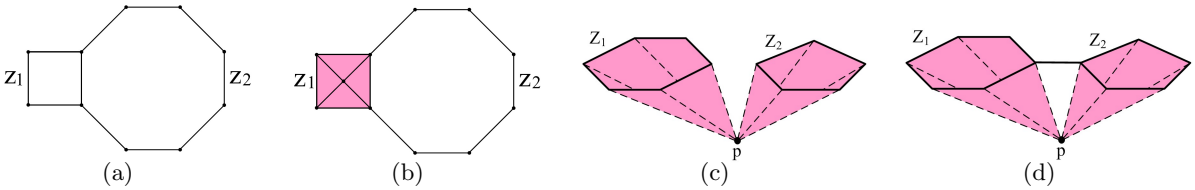


Figure 3: (a,b) the original complex K and the augmented complex K' after destroying the smallest class, $[z_1]$. (c) If the original complex K consists of the two cycles z_1 and z_2, destroying a larger class $[z_1]+[z_2]$ will make all other classes trivial too. (d) The original complex K consists of the two cycles and an edge connecting them. Destroying $[z_1]+[z_2]$ will make all other classes trivial and create a new class.

This lemma leads to the correctness of our algorithm, namely, Theorem 3.3. We prove this theorem by showing that the procedure Measure-All(K) produces the same result as the naive greedy algorithm.

Theorem 3.3. *The procedure* Measure-All(K) *computes* $\mathcal{H}_d$.

4. An Improvement Using Finite Field Linear Algebra

In this section, we present an improvement on the algorithm presented in the previous section, more specifically, an improvement on computing the smallest geodesic ball carrying any nontrivial class (the procedure Bmin). The idea is based on the finite field linear algebra behind the homology.

We first observe that for neighboring vertices, p_1 and p_2, the birth times of the first essential homology class using f_{p_1} and f_{p_2} as filter functions are close (Theorem 4.1). This observation suggests that for each p, instead of computing $B_p^{r(p)}$, we may just test whether the geodesic ball centered at p with a certain radius carries any essential homology class. Second, with some algebraic insight, we reduce the problem of testing whether a geodesic ball carries any essential homology class to the problem of comparing dimensions of two vector spaces. Furthermore, we use Theorem 4.2 to reduce the problem to rank computations of sparse matrices on the $\mathbb{Z}_2$ field, for which we have ready tools from the literature. In what follows, we assume that K has a single component; multiple components can be accommodated with a simple modification.

Complexity. In doing so, we improve the complexity to $O(\beta_d^4 n^3 \log^2 n)$. More detailed complexity analysis is omitted due to space limitations.[3]

Next, we present details of the improvement. In Section 4.1, we prove Theorem 4.1 and provide details of the improved algorithm. In Section 4.2, we explain how to test whether a certain subcomplex carries any essential homology class of K. For convenience, in this section, we use "carrying nonbounding cycles" and "carrying essential homology classes"

[3]This complexity is close to that of the persistent homology algorithm, whose complexity is $O(n^3)$. Given the nature of the problem, it seems likely that the persistence complexity is a lower bound. If this is the case, the current algorithm is nearly optimal. Cohen-Steiner et al.[8] provided a linear algorithm to maintain the persistences while changing the filter function. While interesting, this algorithm is not applicable in our case.

interchangeably, because a geodesic ball carries essential homology classes of K if and only if it carries nonbounding cycles of K.

4.1. The Stability of Persistence Leads to An Improvement

Cohen-Steiner et al.[6] proved that the change, suitably defined, of the persistence of homology classes is bounded by the changes of the filter functions. Since the filter functions of two neighboring vertices, f_{p_1} and f_{p_2}, are close to each other, the birth times of the first nonbounding cycles in both filters are close as well. This leads to Theorem 4.1. A simple proof is provided.

Theorem 4.1. *If two vertices p_1 and p_2 are neighbors, the birth times of the first nonbounding cycles for filter functions f_{p_1} and f_{p_2} differ by no more than 1.*

Proof. p_1 and p_2 are neighbors implies that for any point q, $f_{p_2}(q) \leq f_{p_2}(p_1) + f_{p_1}(q) = 1 + f_{p_1}(q)$, in which the inequality follows the triangular inequality. Therefore, $B_{p_1}^{r(p_1)}$ is a subset of $B_{p_2}^{r(p_1)+1}$. The former carries nonbounding cycles implies that the latter does too, and thus $r(p_2) \leq r(p_1) + 1$. Similarly, we have $r(p_1) \leq r(p_2) + 1$. ∎

This theorem suggests a way to avoid computing $B_p^{r(p)}$ for all $p \in \text{vert}(K)$ in the procedure Bmin. Since our objective is to find the minimum of the $r(p)$'s, we do a breadth-first search through all the vertices with global variables r_{min} and p_{min} recording the smallest $r(p)$ we have found and its corresponding center p, respectively. We start by applying the persistent homology algorithm on K with filter function f_{p_0}, where p_0 is an arbitrary vertex of K. Initialize r_{min} as the birth time of the first nonbounding cycle of K, $r(p_0)$, and p_{min} as p_0. Next, we do a breadth-first search through the rest vertices. For each vertex $p_i, i \neq 0$, there is a neighbor p_j we have visited (the parent vertex of p_i in the breath-first search tree). We know that $r(p_j) \geq r_{min}$ and $r(p_i) \geq r(p_j) - 1$ (Theorem 4.1). Therefore, $r(p_i) \geq r_{min} - 1$. We only need to test whether the geodesic ball $B_{p_i}^{r_{min}-1}$ carries any nonbounding cycle of K. If so, r_{min} is decremented by one, and p_{min} is updated to p_i. After all vertices are visited, p_{min} and r_{min} give us the ball we want.

However, testing whether the subcomplex $B_{p_i}^{r_{min}-1}$ carries any nonbounding cycle of K is not as easy as computing nonbounding cycles of the subcomplex. A nonbounding cycle of $B_p^{r_{min}-1}$ may not be nonbounding in K as we require. For example, in Figure 4(a) and 4(b), the simplicial complex K is a torus with a tail. The pink geodesic ball in the first figure does not carry any nonbounding cycle of K, although it carries its own nonbounding cycles. The geodesic ball in the second figure is the one that carries nonbounding cycles of K. Therefore, we need algebraic tools to distinguish nonbounding cycles of K from those of the subcomplex $B_{p_i}^{r_{min}-1}$.

4.2. Procedure Contain-Nonbounding-Cycle: Testing Whether a Subcomplex Carries Nonbounding Cycles of K

In this section, we present the procedure for testing whether a subcomplex K_0 carries any nonbounding cycle of K. A chain in K_0 is a cycle if and only if it is a cycle of K. However, solely from K_0, we are not able to tell whether a cycle carried by K_0 bounds or not in K. Instead, we write the set of cycles of K carried by K_0, $\mathsf{Z}_d^{K_0}(K)$, and the set of

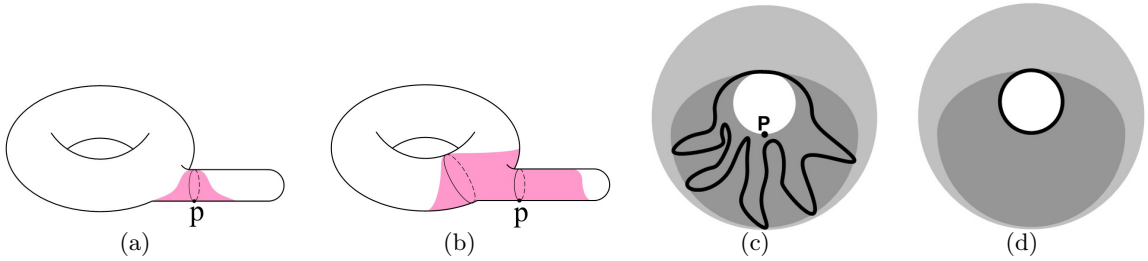


Figure 4: (a,b) In a torus with a tail, only the ball in the second figure carries nonbounding cycles of K, although in both figures the balls have nontrivial topology. (c,d) The cycles with the minimal radius and the minimal diameter, z_r and z_d (Used in Section 5).

boundaries of K carried by K_0, $\mathsf{B}_d^{K_0}(K)$, as sets of linear combinations with certain constraints. Consequently, we are able to test whether any cycle carried by K_0 is nonbounding in K by comparing their dimensions. Formally, we define $\mathsf{B}_d^{K_0}(K) = \mathsf{B}_d(K) \cap \mathsf{C}_d(K_0)$ and $\mathsf{Z}_d^{K_0}(K) = \mathsf{Z}_d(K) \cap \mathsf{C}_d(K_0)$.

Let $\hat{H}_d = [z_1, ..., z_{\beta_d}]$ be the matrix whose column vectors are arbitrary β_d nonbounding cycles of K which are not homologous to each other. The boundary group and the cycle group of K are column spaces of the matrices ∂_{d+1} and $\hat{Z}_d = [\partial_{d+1}, \hat{H}_d]$, respectively. Using finite field linear algebra, we have the following theorem, whose proof is omitted due to space limitations.

Theorem 4.2. *K_0 carries nonbounding cycles of K if and only if*

$$\operatorname{rank}(\hat{Z}_d^{K\setminus K_0}) - \operatorname{rank}(\partial_{d+1}^{K\setminus K_0}) \neq \beta_d.$$

where ∂_{d+1}^i and $\hat{Z}_d^i$ are the i-th rows of the matrices ∂_{d+1} and $\hat{Z}_d$, respectively.

We use the algorithm of Wiedemann[18] for the rank computation. In our algorithm, the boundary matrix ∂_{d+1} is given. The matrix $\hat{H}_d$ can be precomputed as follows. We perform a column reduction on the boundary matrix ∂_d to compute a basis for the cycle group $\mathsf{Z}_d(K)$. We check elements in this basis one by one until we collect β_d of them forming $\hat{H}_d$. For each cycle z in this cycle basis, we check whether z is linearly independent of the d-boundaries and the nonbounding cycles we have already chosen. More details are omitted due to space limitations.

5. Localizing Classes

In this section, we address the localization problem. We formalize the localization problem as a combinatorial optimization problem: Given a simplcial complex K, compute the representative cycle of a given homology class minimizing a certain objective function. Formally, given an objective function defined on all the cycles, $\text{cost} : \mathsf{Z}_d(K) \to \mathbb{R}$, we want to localize a given class with its *optimally localized cycle*, $z_{opt}(h) = \operatorname{argmin}_{z\in h} \text{cost}(z)$. In general, we assume the class h is given by one of its representative cycles, z_0.

We explore three options of the objective function $\text{cost}(z)$, i.e. the *volume*, *diameter* and *radius* of a given cycle z. We show that the cycle with the minimal volume and the cycle with the minimal diameter are NP-hard to compute. The cycle with the minimal radius,

which is the localized-cycle we defined and computed in previous sections, is a fair choice. Due to space limitations, we omit proofs of theorems in this section.

Definition 5.1 (Volume)**.** The volume of z is the number of its simplices, $\mathrm{vol}(z) = \mathrm{card}(z)$.

For example, the volume of a 1-dimensional cycle, a 2-dimensional cycle and a 3-dimensional cycle are the numbers of their edges, triangles and tetrahedra, respectively. A cycle with the smallest volume, denoted as z_v, is consistent to a "well-localized" cycle in intuition. Its 1-dimensional version, the shortest cycle of a class, has been studied by researchers [14, 19, 11]. However, we prove in Theorem 5.2 that computing z_v of h is NP-hard.[4] The proof is by reduction from the NP-hard problem MAX-2SAT-B [17]. More generally, we can extend the the volume to be the sum of the weights assigned to simplices of the cycle, given an arbitrary weight function defined on all the simplices of K. The corresponding smallest cycle is still NP-hard to compute.

Theorem 5.2. *Computing z_v for a given h is NP-hard.*

When it is NP-hard to compute z_v, one may resort to the geodesic distance between elements of z. The second choice of the objective function is the diameter.

Definition 5.3 (Diameter)**.** The diameter of a cycle is the diameter of its vertex set, $\mathrm{diam}(z) = \mathrm{diam}(\mathrm{vert}(z))$, in which the diameter of a set of vertices is the maximal geodesic distance between them, formally, $\mathrm{diam}(S) = \max_{p,q \in S} \mathrm{dist}(p, q)$.

Intuitively, a representative cycle of h with the minimal diameter, denoted z_d, is the cycle whose vertices are as close to each other as possible. The intuition will be further illustrated by comparison against the radius criterion. We prove in Theorem 5.4 that computing z_d of h is NP-hard, by reduction from the NP-hard *Multiple-Choice Cover Problem* (MCCP) of Arkin and Hassin [2].

Theorem 5.4. *Computing z_d for a given h is NP-hard.*

The third option of the objective function is the radius.

Definition 5.5 (Radius)**.** The radius of a cycle is the radius of the smallest geodesic ball carrying it, formally, $\mathrm{rad}(z) = \min_{p \in \mathrm{vert}(K)} \max_{q \in \mathrm{vert}(z)} \mathrm{dist}(p, q)$, where $\mathrm{vert}(K)$ and $\mathrm{vert}(z)$ are the sets of vertices of the given simplicial complex K and the cycle z, respectively.

The representative cycle with the minimal radius, denoted as z_r, is the same as the localized-cycle defined and computed in previous sections. Intuitively, z_r is the cycle whose vertices are as close to a vertex of K as possible. However, z_r may not necessarily be localized in intuition. It may wiggle a lot while still being carried by the smallest geodesic ball carrying the class. See Figure 4(c), in which we localize the only nontrivial homology class of an annulus (the light gray area). The dark gray area is the smallest geodesic ball carrying the class, whose center is p. Besides, the cycle with the minimal diameter (Figure 4(d)) avoids this wiggling problem and is concise in intuition. This in turn justifies the choice of diameter.[5] We can prove that z_r can be computed in polynomial time and is a 2-approximation of z_d.

[4] Erickson and Whittlesey [14] localized 1-dimensional classes with their shortest representative cycles. Their polynomial algorithm can only localize classes in the shortest homology basis, not arbitrary given classes.

[5] This figure also illustrates that the radius and the diameter of a cycle are not strictly related. For the cycle z_r in the left, its diameter is twice of its radius. For the cycle z_d in the center, its diameter is equal to its radius.

Theorem 5.6. *We can compute z_r in polynomial time.*

Theorem 5.7. $\mathrm{diam}(z_r) \leq 2\,\mathrm{diam}(z_d)$.

This bound is a tight bound. In Figure 4(c) and 4(d), the diameter of the cycle z_r is twice of the radius of the dark gray geodesic ball. The diameter of the cycle z_d is the same as the radius of the ball. We have $\mathrm{diam}(z_r) = 2\,\mathrm{diam}(z_d)$.

Acknowledgements

The authors wish to acknowledge constructive comments from anonymous reviewers and fruitful discussions on persistent homology with Professor Herbert Edelsbrunner.

References

[1] P. K. Agarwal, H. Edelsbrunner, J. Harer, and Y. Wang. Extreme elevation on a 2-manifold. *Discrete & Computational Geometry*, 36:553–572, 2006.
[2] E. M. Arkin and R. Hassin. Minimum-diameter covering problems. *Networks*, 36(3):147–155, 2000.
[3] G. Carlsson. Persistent homology and the analysis of high dimensional data. Symposium on the Geometry of Very Large Data Sets, Febrary 2005. Fields Institute for Research in Mathematical Sciences.
[4] G. Carlsson, T. Ishkhanov, V. de Silva, and L. J. Guibas. Persistence barcodes for shapes. *International Journal of Shape Modeling*, 11(2):149–188, 2005.
[5] C. Carner, M. Jin, X. Gu, and H. Qin. Topology-driven surface mappings with robust feature alignment. In *IEEE Visualization*, p. 69, 2005.
[6] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37:103–120, 2007.
[7] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Extending persistent homology using Poincaré and Lefschetz duality. *Foundations of Computational Mathematics*, to appear.
[8] D. Cohen-Steiner, H. Edelsbrunner, and D. Morozov. Vines and vineyards by updating persistence in linear time. In *Symposium on Computational Geometry*, pp. 119–126, 2006.
[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.
[10] V. de Silva and R. Ghrist. Coverage in sensor networks via persistent homology. *Algebraic & Geometric Topology*, 2006.
[11] T. K. Dey, K. Li, and J. Sun. On computing handle and tunnel loops. In *IEEE Proc. NASAGEM*, 2007.
[12] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.
[13] J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. *Discrete & Computational Geometry*, 31(1):37–59, 2004.
[14] J. Erickson and K. Whittlesey. Greedy optimal homotopy and homology generators. In *SODA*, pp. 1038–1046, 2005.
[15] R. Ghrist. Barcodes: the persistent topology of data. Amer. Math. Soc Current Events Bulletin.
[16] J. R. Munkres. *Elements of Algebraic Topology.* Addison-Wesley, Redwook City, California, 1984.
[17] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proc. 20th ACM Symposium on Theory of computing*, pp. 229–234, New York, NY, USA, 1988. ACM Press.
[18] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54–62, 1986.
[19] Z. J. Wood, H. Hoppe, M. Desbrun, and P. Schröder. Removing excess topology from isosurfaces. *ACM Trans. Graph.*, 23(2):190–208, 2004.
[20] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.
[21] A. Zomorodian and G. Carlsson. Localized homology. In *Shape Modeling International*, pp. 189–198, 2007.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 181-192
www.stacs-conf.org

SHORTEST VERTEX-DISJOINT TWO-FACE PATHS IN PLANAR GRAPHS

ÉRIC COLIN DE VERDIÈRE [1] AND ALEXANDER SCHRIJVER [2]

[1] Laboratoire d'informatique
École normale supérieure, CNRS
45, rue d'Ulm
75005 Paris
France
E-mail address: Eric.Colin.de.Verdiere@ens.fr
URL: http://www.di.ens.fr/~colin/

[2] Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
Netherlands
E-mail address: lex@cwi.nl
URL: http://homepages.cwi.nl/~lex/

Abstract. Let G be a directed planar graph of complexity n, each arc having a non-negative length. Let s and t be two distinct faces of G; let $s_1, \ldots, s_k$ be vertices incident with s; let $t_1, \ldots, t_k$ be vertices incident with t. We give an algorithm to compute k pairwise vertex-disjoint paths connecting the pairs (s_i, t_i) in G, with minimal total length, in $O(kn \log n)$ time.

1. Introduction

The *vertex-disjoint paths problem* is described as follows: given any (directed or undirected) graph and k pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices, find k pairwise vertex-disjoint paths connecting the pairs (s_i, t_i), if they exist. This problem is well-known also because of its motivation by VLSI-design.

For a fixed number k of pairs of terminals, this problem is polynomial-time solvable in a directed planar graph, as shown by Schrijver [Sch94], and in any undirected graph, as shown by Robertson and Seymour [RS95]. However, Raghavan [Rag86] and Kramer and van Leeuwen [KvL84] proved that it is NP-hard when k is not fixed, even on a planar undirected

1998 ACM Subject Classification: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical algorithms and problems—*Computations on discrete structures; routing and layout*; G.2.2 [Mathematics of Computing]: Graph theory—*Graph algorithms; network problems; path and circuit problems*.

Key words and phrases: algorithm, planar graph, disjoint paths, shortest path.

Most of this work was done while the first author was visiting the second author at CWI Amsterdam.

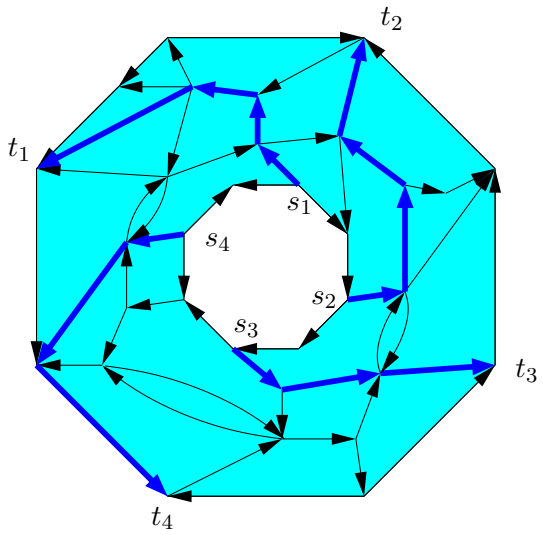


Figure 1: An instance of the problem and a solution (in bold lines).

graph; it belongs to the more general class of *integer multicommodity flow problems* [Sch03, Chapter 70], many variants of which are NP-hard.

If the graph is planar, two special cases are solvable in time linear in the complexity of the graph, even if k is not fixed:

(a) if all terminals lie on the outer face, as proved by Suzuki et al. [SAN90];
(b) if the terminals $s_1, \dots, s_k$ are incident with a common face s, the terminals $t_1, \dots, t_k$ are incident with a common face t, and the faces s and t are distinct, as proved by Ripphausen-Lipa et al. [RLWW96].

In this paper, we consider a graph where each edge has a nonnegative length, and we wish to solve the vertex-disjoint paths problem using paths with minimal total length. Of course, this is harder than the vertex-disjoint paths problem. In case (a), the problem is known to be solvable in polynomial time (even if k is not fixed) if the cyclic order of the terminals is $s_1, \dots, s_k, t_k, \dots, t_1$ (by reduction to the max-flow problem, after replacing each vertex by two vertices connected by an arc, so that the problem is to find arc-disjoint paths in this new graph) [vdHdP02]. Our goal is to solve the vertex-disjoint paths problem with minimal total length in case (b). We give an algorithm to do this in $O(kn \log n)$ time (see Figure 1):

Theorem 1.1. *Let G be a planar directed graph with n vertices and arcs, each arc having a nonnegative length. Let s and t be two distinct faces of G; let $s_1, \dots, s_k$ be vertices incident with s; let $t_1, \dots, t_k$ be vertices incident with t. Then we can compute k pairwise vertex-disjoint paths connecting the pairs (s_i, t_i) in G, with minimal total length, in $O(kn \log n)$ time.*

The value of k is not fixed in this result. Note that this theorem also holds if G is an undirected graph: simply replace every edge of this graph by two oppositely directed arcs and apply the previous result to this new graph. The same problem for *non-crossing* shortest paths, that is, paths that are allowed to overlap along vertices and edges but not to cross in the plane, is solvable in $O(n \log n)$ time, as shown by Takahashi et al. [TSN96].

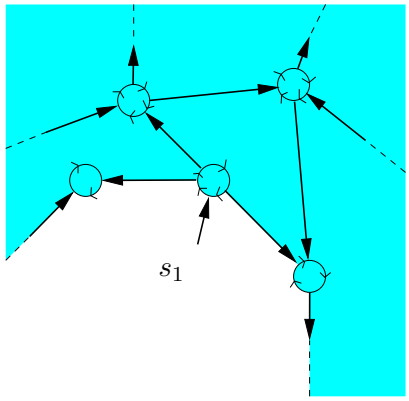


Figure 2: Construction of the graph $D = (W, A)$ from the graph G. The thin arcs on the rings have length zero.

The high-level approach of our algorithm is the following. We first show that we may assume without loss of generality that G satisfies some additional properties and transform G into another planar directed graph D; in this graph, it suffices to solve the same problem for *arc-disjoint* instead of *vertex-disjoint* paths (Section 2). Then we translate our problem in terms of flows in the graph D (Section 3). In Section 4, we introduce the residual graph and state some of its properties that we will use. In Section 5, we explain how to increase the value of an integer flow. By repeated applications of this algorithm, we obtain vertex-disjoint paths in G between the terminals, but they may fail to connect the pairs (s_i, t_i). We show that it suffices to "rotate" the flow a few times to change the connections between the terminals (Section 6) and explain how to do that efficiently (Section 7). A generalization of the notion of potential allows us to assume that all lengths in the residual graph are nonnegative, which makes the algorithm efficient.

2. Preliminaries

We assume that we are given an embedding of the directed graph G in the plane. More precisely, only a *combinatorial embedding* of G is necessary, which means that the cyclic order of the arcs around a vertex is known.

We can assume that G is connected and that t is the outer face of the embedding of G. Up to re-indexing the pairs (s_i, t_i), we may assume that $s_1, \dots, s_k$ and $t_1, \dots, t_k$ are in clockwise order: indeed, if such a reordering does not exist, then there cannot exist vertex-disjoint paths connecting the pairs (s_i, t_i).

We may assume that each terminal vertex has degree one as follows: to each terminal vertex s_i (resp. t_i), attach an arc (of length zero, for example) (s'_i, s_i) (resp. (t_i, t'_i)) inside s (resp. t), where s'_i (resp. t'_i) is a new vertex; use the s'_i and the t'_i as terminals, instead of the s_i and the t_i. Clearly, any solution to the problem in this augmented graph yields a solution in the original graph G.

We transform G into another directed planar graph $D = (W, A)$ by replacing each non-terminal vertex v of G by a small clockwise "ring" of arcs; see Figure 2. Every arc a of D that is on no ring corresponds to an arc of G and its length, $\lambda(a)$, is the length of this arc

in G; it is thus nonnegative. The length $\lambda(a)$ of an arc a on a ring is zero. The function λ is fixed in this whole paper.

An (s,t)-*path* in D or G is a path from some vertex in $\{s_1,\dots,s_k\}$ to some vertex in $\{t_1,\dots,t_k\}$; an (s_i,t_i)-*path* is a path connecting some pair of terminals (s_i,t_i).

Proposition 2.1. *Let P be a minimum-length set of k vertex-disjoint (s_i,t_i)-paths in D. Then P gives, in $O(n)$ time, a minimum-length set of k vertex-disjoint (s_i,t_i)-paths in G. If no such set P exists, then the original problem in G has no solution.*

Proof. Consider such a set of (s_i,t_i)-paths P in D. We claim that a given ring r of D can be used by at most one path in P. Indeed, since s and t are distinct faces, $\mathbf{R}^2 \setminus \{s \cup t\}$ is an annulus. Since the paths in P are vertex-disjoint and connect s to t, every point of the annulus that does not belong to a path in P is on the left of exactly one path and on the right of exactly one path in P. In particular, the center c of r is on the right of exactly one path in P. But every path using r has c on its right, because the arcs of r are oriented clockwise. This proves the claim.

Thus, P corresponds, in G, to k pairwise vertex-disjoint (s_i,t_i)-walks. Removing the loops from these walks in $O(n)$ time does not increase the total length and gives a set of k vertex-disjoint (s_i,t_i)-paths in G.

Conversely, any solution of the original vertex-disjoint problem in G gives a set of k vertex-disjoint paths in D, of the same length, connecting the appropriate pairs of terminals. So the paths obtained in the previous paragraph have minimal total length; furthermore, if no such set of paths P exists, then the problem in G admits no solution. ∎

So we reduced the problem in G to the same problem in the graph D. The point now is that the vertices of D have degree three, except the terminals, which have degree one; because of these degree conditions, a set of *arc*-disjoint (s,t)-paths or circuits in D is actually a set of *vertex*-disjoint (s,t)-paths or circuits in D, so we now have to solve a problem on *arc*-disjoint paths. This enables a flow approach on D, which we will develop in the next section.

3. Flows and winding numbers

In this paper, a *flow* in $D=(W,A)$ is an element $x \in \mathbf{R}^A$ such that:

(1) for each arc $a \in A$, $0 \le x(a) \le 1$;

(2) for each non-terminal vertex v, the following *flow conservation law* holds:

$$\sum_{a \,|\, v=\text{source}(a)} x(a) = \sum_{a \,|\, v=\text{target}(a)} x(a).$$

The *value* of a flow x equals the total flow leaving the vertices $s_1,\dots,s_k$: if a_i is the arc incident with s_i, then the value of x equals $\sum_{i=1}^k x(a_i)$. A *circulation* is a flow of value zero. A *length function* (or *cost function*) κ on D is an element of $\mathbf{R}^A$; λ is a length function. The *length* (or *cost*) of a flow x with respect to κ is defined to be $\kappa^\top x$.

An *integer* flow is a flow in $\{0,1\}^A$; it is a set of arc-disjoint (s,t)-paths and circuits in D. Actually, by the degree conditions on D, it is a set of *vertex-disjoint* (s,t)-paths and circuits.

Let A^{-1} be the set of arcs in A with reverse orientation. If $\kappa \in \mathbf{R}^A$ is a length function, we define the length of an arc $a^{-1} \in A^{-1}$ to be $\kappa(a^{-1}) = -\kappa(a)$.

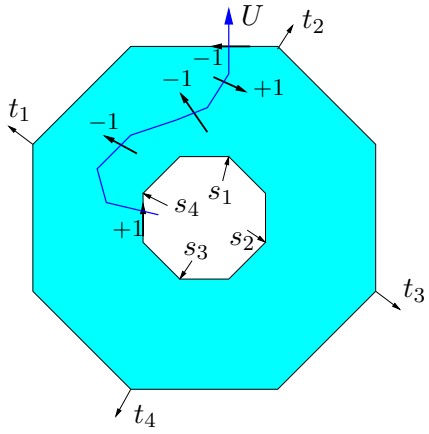


Figure 3: The path U in the dual graph D^* and the corresponding value of u on the arcs of D. Only the non-zero values of u are indicated, on the arcs in bold lines. Here $m = m_s - m_t = 3 - 1 = 2$.

Let $X \in \mathbf{R}^{A \cup A^{-1}}$; we define $z^X \in \mathbf{R}^A$ by $z^X(a) = X(a) - X(a^{-1})$. If γ is a walk in $(W, A \cup A^{-1})$, by a slight abuse of notation, we define z^γ to be z^X, where $X(a)$ (resp. $X(a^{-1})$) is the number of times γ travels through the arc a (resp. a^{-1}). The length of γ with respect to a length function κ is thus $\kappa^\top z^\gamma$.

We now want to take into account how a flow "turns around" the inner face s of G. To do this, consider the (undirected) dual graph D^* of D, that is, the planar graph that has one vertex f^* inside each face f of D and such that f_1^* and f_2^* are connected by an edge e^* if and only if f_1 and f_2 are separated by an arc e in D; in that case, e^* crosses e but no other arc of D. Let U be a path (fixed in this whole paper) from s^* to t^* in D^* (Figure 3). For each arc a in A, define $u(a)$ to be 0 if a does not cross U, $+1$ if a crosses U from left to right, and -1 if a crosses U from right to left. This defines an element $u \in \mathbf{R}^A$. The *winding number* of a flow x equals $u^\top x$, the value of the flow through u counted algebraically. Also, for any $X \in \mathbf{R}^{A \cup A^{-1}}$, the winding number of X is $u^\top z^X$.

Let $m_s \in [1, k]$ be such that the first arc of U is, in the cyclic order around the face s, between s_{m_s} and $s_{m_s+1 \bmod k}$. Similarly, let m_t be such that the last arc of U is between t_{m_t} and $t_{m_t+1 \bmod k}$. Let $m = m_s - m_t$.

The following lemma will be used repeatedly.

Lemma 3.1. *Let γ be any circuit in $(W, A \cup A^{-1})$. Then the winding number of γ belongs to $\{-1, 0, +1\}$. If γ encloses s in the plane, then it has winding number $+1$ if it is clockwise and -1 if it is counter-clockwise. Otherwise, γ has winding number 0.*

Proof. This is a consequence of the Jordan curve theorem. The winding number of γ is the number of times the path U crosses γ from the right to the left, minus the number of times U crosses γ from the left to the right. Assume γ is clockwise, the other case being analogous. The winding number of γ is the number of times U exits the region enclosed by γ minus the number of times it enters this region.

If γ does not enclose s, then both endpoints of U are outside γ, so the winding number is zero. If γ encloses s, the source of U is inside the region enclosed by γ while its target is outside, so the winding number is $+1$. ∎

We can now reformulate our arc-disjoint paths problem in D in terms of flows in D:

Proposition 3.2. *Let x be an integer flow in D of value k with minimal cost subject to the condition that its winding number, modulo k, equals m. Then x gives, in $O(n)$ time, k vertex-disjoint (s_i, t_i)-paths in D of minimal total length. If there exists no such flow, then there does not exist k vertex-disjoint (s_i, t_i)-paths in D.*

Proof. As noted above, the degree conditions on D imply that the flow x is a set of *vertex*-disjoint (s,t)-paths or circuits in D. Let γ be a circuit in x. If γ has non-zero winding number, then γ separates s and t, which implies that x has value zero, a contradiction. If γ has winding number zero, then removing it from x yields another flow with the same properties. Since we can remove such circuits in $O(n)$ time, we may assume that x contains only (s,t)-paths. By the assumption on the winding number, these paths connect the pairs (s_i, t_i), for $i = 1, \ldots, k$.

Furthermore, any k vertex-disjoint (s_i, t_i)-paths in D correspond to a flow in D of value k and of winding number equal, modulo k, to m. It follows that the paths obtained have minimal total length. ∎

By Propositions 2.1 and 3.2, to prove Theorem 1.1, it suffices to show that we can, in $O(kn \log n)$ time, find an integer flow in D of value k and with minimal cost subject to the condition that its winding number, modulo k, equals m.

4. The residual graph

In this section, we introduce the *residual graph* of D in the special case of integer flows; it is a classical tool for dealing with maximal flows and flows of minimal cost [Sch03, Chapters 10–12].

Let x be an integer flow on $D = (W, A)$. Let A_x be the subset of $A \cup A^{-1}$ defined by

$$A_x = \{a \mid x(a) = 0\} \cup \{a^{-1} \mid x(a) = 1\}.$$

The *residual graph* of D with respect to x is the directed graph $D_x = (W, A_x)$; it is thus the graph obtained from D by reversing the sign of the length and winding number and the orientation of the arcs a such that $x(a) = 1$.

The following lemma explains the interest of the residual graph; the first two assertions are well-known.

Lemma 4.1. *Let x be an integer flow in D.*

(i) *D_x has no (s,t)-path if and only if x has maximal value in D among all flows.*
(ii) *Assume that x has maximal value in D; let κ be a length function. Then D_x has no negative-length directed circuit with respect to κ if and only if x has minimal cost, with respect to κ, among all flows in D with the same value.*
(iii) *Assume x has maximal value in D. Then D_x has no directed circuit with winding number one if and only if x has maximal winding number among all flows in D with the same value.*

Proof. In these three assertions, the "if" part is easy: If D_x has an (s,t)-path or circuit γ, then, by construction of D_x, $y := x + z^\gamma$ is an integer flow in D; its cost equals the cost of x in D plus the cost of γ in D_x; its winding number equals the winding number of x plus the winding number of γ; and its value equals the value of x plus one if γ is a path, or the value of x if γ is a circuit.

Conversely, let x be an integer flow in D and let y be any flow in D. Consider $y - x$ in the graph D. By construction of D_x, this is a flow in D_x, in the sense that the flow conservation law holds at each vertex of D (except at the terminals) and that, for each arc $a \in A$, we have $(y - x)(a) \geq 0$ if $a \in A_x$ and $(y - x)(a) \leq 0$ if $a^{-1} \in A_x$. In particular, $y - x$ can be written as $\sum_{\gamma \in Z} \alpha_\gamma z^\gamma$, where Z is a set of (s,t)-paths, (t,s)-paths, and circuits in D_x, and the α_γ are positive real numbers.

Now, to prove the "only if" part of (i), simply note that, if D_x has no (s,t)-path, then there is no (s,t)-path in Z; thus, the value of y cannot be greater than the value of x. To prove the "only if" part of (ii) and (iii), assume that x and y both have maximal value in D. Then, by (i), Z contains no (s,t)-path, hence also no (t,s)-path, hence only circuits. If D_x has no negative-length directed circuit, the cost of y is at least the cost of x; this proves (ii). If D_x has no directed circuit with winding number one, then y cannot have winding number higher than x, for otherwise $y - x$ would contain at least one circuit with positive winding number, hence with winding number one (Lemma 3.1). This proves (iii). ∎

A length function κ is *nonnegative on* D_x if κ is nonnegative on every arc in A_x; that is, for each $a \in A$, $\kappa(a) \geq 0$ if $x(a) = 0$ and $\kappa(a) \leq 0$ if $x(a) = 1$.

5. Increasing the flow in D

In this section, we explain how to compute a minimum-cost flow in D in $O(kn \log n)$ time. The algorithm uses only very classical minimum-cost flow techniques, but we indicate it for completeness and because Section 7 will use some similar ideas.

Let $p \in \mathbf{Z}$. A *p-flow* is an *integer* flow in D of value p. Let κ and κ' be two length functions on D; we write $\kappa \simeq \kappa'$ if $\kappa^\top z^\gamma = \kappa'^\top z^\gamma$ for each closed walk γ in $(W, A \cup A^{-1})$. (This notion is equivalent to the notion of potential.)

Lemma 5.1. *Let $\kappa \simeq \kappa'$. Then any minimum-cost k-flow with respect to κ is also a minimum-cost k-flow with respect to κ'.*

Proof. By Lemma 4.1(ii), a k-flow x has minimum cost with respect to κ if and only if D_x has no negative-length circuit with respect to κ. Since $\kappa \simeq \kappa'$, circuits in D_x have the same length with respect to κ and to κ'. ∎

The following result follows from classical minimum-cost flow techniques.

Lemma 5.2. *Let x be a p-flow in D and let κ be a length function that is nonnegative on D_x. Then, in $O(n \log n)$ time, we can find a $(p+1)$-flow x' and a length function $\kappa' \simeq \kappa$ that is nonnegative on $D_{x'}$, unless x has maximal value.*

Proof. We temporarily add to D_x two vertices s and t, and arcs (s, s_i) and (t_i, t) of length zero, for $i = 1, \ldots, k$. Let D'_x be the resulting graph. We compute a shortest path tree of D'_x with root s, with respect to κ, in $O(n \log n)$ time using Dijkstra's algorithm [Dij59] speeded up with Fibonacci heaps [FT87], because all lengths are nonnegative[1]. If there is no path from s to t in D'_x, then D_x has no (s,t)-path, hence, by Lemma 4.1(i), x has maximal value.

Otherwise, for each vertex v of D'_x, let $d(v)$ be the distance from s to v with respect to κ, as computed by Dijkstra's algorithm above. For each arc $a = (u, v)$ of A_x, we have

[1]We could do that in $O(n)$ time using the algorithm by Henzinger et al. [HKRS97], but that would not change the asymptotic complexity of the entire algorithm.

$d(v) \leq d(u) + \kappa(a)$ by the triangle inequality, with equality if a is on the shortest path tree. For each arc $a = (u, v)$ of A_x, let $\kappa'(a) = \kappa(a) + d(u) - d(v)$; clearly, $\kappa' \simeq \kappa$. We have $\kappa'(a) \geq 0$, and $\kappa'(a) = 0$ if a is on the shortest path tree. Let γ be the (s,t)-path in D_x corresponding to the path from s to t in D'_x in the shortest path tree. Now, let $x' = x + z^\gamma$; since κ' is nonnegative on the arcs of D_x and is zero on the arcs of γ, it is nonnegative on $D_{x'}$. ∎

Starting with the zero flow x (for which $D_x = D$) and the length function $\kappa = \lambda$, we repeatedly apply Lemma 5.2. We obtain a flow x_0 with maximal value p and a length function $\kappa_0 \simeq \lambda$ such that κ_0 is nonnegative on D_{x_0}. This takes $O(pn \log n) = O(kn \log n)$ time. If $p < k$, then the original problem has no solution, hence we stop here. Otherwise, Lemmas 4.1(i) and 5.1 imply that x_0 is a minimum-cost k-flow with respect to λ. Let w_0 be the winding number of x_0. If $w_0 \equiv m \pmod k$, then we are done by Propositions 2.1 and 3.2; so we henceforth assume $w_0 \not\equiv m \pmod k$.

6. Finding the winding number

A (k, w)-*flow* is an integer flow in D of value k and winding number w. Let w_1 and w_2 be the integers equal, modulo k, to m that are the closest to w_0 and satisfy $w_1 < w_0 < w_2$. The following proposition states that the problem boils down to finding minimum-cost (k, w)-flows, for $w = w_1$ and $w = w_2$:

Proposition 6.1. *There is a minimum-cost integer flow in D (with respect to λ) of value k and winding number equal, modulo k, to m that is either a (k, w_1)-flow or a (k, w_2)-flow.*

Proof. For every integer w, let μ_w be the minimal cost of the (k, w)-flows. (It is infinite if no (k, w)-flow exists.) By Lemma 4.1(iii), the set $\{w \mid \mu_w < \infty\}$ is an interval of integers.

We show that for every integer w such that μ_{w-1}, μ_w, and μ_{w+1} are finite, we have

$$2\mu_w \leq \mu_{w-1} + \mu_{w+1}. \tag{6.1}$$

Indeed, let x and x' be minimum-cost $(k, w-1)$- and $(k, w+1)$-flows, respectively. Then $x' - x$ gives a nonnegative integer circulation in D_x of winding number 2, i.e., a flow y of value zero in D such that, for each $a \in A$, $y(a) \geq 0$ if $a \in A_x$ and $y(a) \leq 0$ if $a^{-1} \in A_x$. So the support of $x' - x$ contains a directed circuit γ in D_x of positive winding number, hence 1. Then $x + z^\gamma$ and $x' - z^\gamma$ are both (k, w)-flows. Thus

$$2\mu_w \leq \lambda^\top (x + z^\gamma) + \lambda^\top (x' - z^\gamma) = \lambda^\top x + \lambda^\top x' = \mu_{w-1} + \mu_{w+1},$$

which proves (6.1).

So μ_w is monotonically non-increasing for $w \leq w_0$ and monotonically non-decreasing for $w \geq w_0$. Thus Proposition 6.1 holds. ∎

7. Rotating the flow in D

Let κ and κ' be two length functions on D; we write $\kappa \sim \kappa'$ if $\kappa^\top z^\gamma = \kappa'^\top z^\gamma$ for each closed walk γ *with winding number zero* in $(W, A \cup A^{-1})$. Clearly, $\kappa \simeq \kappa'$ implies $\kappa \sim \kappa'$.

Proposition 7.1. *Let $\kappa \sim \kappa'$. Then any minimum-cost (k, w)-flow with respect to κ is also a minimum-cost (k, w)-flow with respect to κ'.*

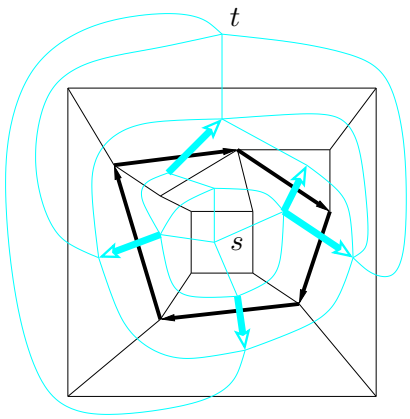


Figure 4: Illustration of Lemma 7.2: A minimal cut in H^* corresponds to a circuit with winding number one in H. The primal graph H is depicted in black lines, with thicker lines for the arcs of the circuit. The dual graph H^* is depicted in light color, with thicker lines for the arcs of the cut.

Proof. Let x and y be two (k, w)-flows in D. Then $y - x$ is a circulation in $(W, A \cup A^{-1})$, i.e., a sum of terms of the form z^γ, where γ is a circuit in $(W, A \cup A^{-1})$. Furthermore, there are as many circuits with winding number $+1$ as with winding number -1 in this sum.

We have $(\kappa' - \kappa)^\top z^\gamma = 0$ for every such circuit with winding number zero. Moreover, if γ has winding number $+1$ and γ' has winding number -1, it follows from the definition of "$\sim$" that $\kappa^\top(z^\gamma + z^{\gamma'}) = \kappa'^\top(z^\gamma + z^{\gamma'})$. We thus have $\kappa^\top(y - x) = \kappa'^\top(y - x)$, implying the result. ∎

We view D as an undirected planar graph H; s and t are two faces of H. Let H^* be its dual graph. If e is an *oriented* edge of H, then e^* is the dual edge oriented so that e^* crosses e from right to left.

A *cut* of H^* is a set X^* of *oriented* edges of H^* such that any directed path from s^* to t^* uses at least one oriented edge of X^*. The following lemma is inspired by Reif [Rei83, Propositions 1 and 2]. See Figure 4.

Lemma 7.2. *Let X be a set of oriented edges of H. Then X contains the oriented edges of some circuit with winding number one in H if and only if X^* is a cut of H^*.*

Proof. If we have a directed circuit γ with winding number one, then its dual is a cut. Indeed, consider an (s^*, t^*)-path π in H^*. The face s belongs to the interior of γ, while the face t belongs to the exterior of γ; let e^* be the first oriented edge of π that crosses γ; its source is inside γ while its target is outside γ. By our choice of orientation, e belongs to γ.

Conversely, let X^* be a cut of H^*; we will prove that X contains a circuit with winding number one. Without loss of generality, we may assume that X^* is a cut that is *minimal* with respect to inclusion.

First, label "S" a face f of H if there is, in H^*, a path from s^* to f^* that does not use any oriented edge of X^*. Similarly, label "T" a face f of H if there is, in H^*, a path from f^* to t^* that does not use any oriented edge of X^*. Since X^* is a cut, no face of H is labeled both "S" and "T". We claim that X is precisely the set of oriented edges of H whose right face is labeled "S" and whose left face is labeled "T". Clearly, such edges must belong to X. Conversely, let e be an oriented edge of X; by minimality of X, there is an

(s^*, t^*)-path in H^* that avoids $(X \setminus e)^*$ and uses e^* exactly once. Thus the source of e^* is reachable from s^* without using any oriented edge of X^*, and t^* is reachable from the target of e^* without using any oriented edge of X^*. This proves the claim. In particular, every face of H is labeled either "S" or "T".

Let S be the subset of the plane made of the faces labeled "S", together with the *open* edges whose both incident faces are labeled "S". Similarly, let T be the union of the faces labeled "T" together with the open edges whose both incident faces are labeled "T". By the previous paragraph, S and T are disjoint subsets of the plane, and they are connected. Let v be a vertex of H. We claim that there cannot be four faces incident with v, in this cyclic order around v, that belong respectively to S, T, S, and T. This follows from the Jordan curve theorem: assume that we have such faces. Then, by connectivity of S, there is a simple closed curve in $S \cup \{v\}$ that goes through v and has faces of T on both sides of it at v. This curve does not intersect T and separates T, contradicting its connectivity.

The two previous paragraphs together imply that either X has no edge incident with v, or X has exactly one oriented edge whose target is v and one oriented edge whose source is v. Thus X is a union of vertex-disjoint circuits. Let γ be such a circuit; since S and T are connected, and since the faces on the left (resp. right) of γ are in T (resp. S), γ has winding number one. Hence X contains a circuit with winding number one. ∎

Proposition 7.3. *Let x be a (k, w)-flow in D and let κ be a length function that is nonnegative on D_x. Then, in $O(n \log n)$ time, we can find a $(k, w+1)$-flow x' and a length function $\kappa' \sim \kappa$ that is nonnegative on $D_{x'}$, unless there is no (k, w')-flow with $w' > w$.*

Proof. Let e be an oriented edge of H; if e corresponds to an arc a of A_x, then we define the length of e in H to be $\kappa(a) \geq 0$; otherwise, we define the length of e to be ∞. So a walk in D_x corresponds to a walk in H of the same length, and a walk in H corresponds to a walk in D_x if and only if it has finite length. Define the *capacity* $c(e^*)$ of an oriented edge e^* of H^* to be the length of e.

We can detect in $O(n)$ time whether the oriented edges of finite capacity constitute a cut in H^*. If this is not the case, then every cut must use an oriented edge of infinite capacity, hence, by Lemma 7.2, D_x has no circuit of winding number one. It follows that x has maximal winding number among all k-flows, by Lemma 4.1(iii). Otherwise, we compute a minimal cut in H^*, which corresponds to a shortest circuit with winding number one in D_x, as follows.

A *flow* in H^* is a function φ that associates, to each oriented edge e^* of H^*, a real number that is nonnegative and no greater than $c(e^*)$, such that the flow conservation law holds at each vertex of H^* except at s^* and t^*. The *value* of φ is the total flow leaving s^*.

In $O(n \log n)$ time, we compute a flow φ of maximal value in H^* with respect to these capacities, using the algorithm by Borradaile and Klein [BK06]. It is well-known, by the "max-flow min-cut" theorem [Sch03, Theorem 10.3], that φ corresponds to a cut of minimal cost in H^*: the cut is the set of oriented edges that leave the set of vertices reachable from s^* by using only oriented edges e^* of H^* such that $\varphi(e^*) < c(e^*)$ or $\varphi(e^{*-1}) > 0$.

Such a cut X^* can be computed in $O(n)$ time. Moreover, by replacing all the zero capacities in H^* by infinitesimally small capacities before applying the maximal flow algorithm, we may assume that X^* is a cut that is minimal with respect to inclusion. By Lemma 7.2, we thus obtain a circuit γ of winding number one that has minimal length in D_x.

For each arc a of $A \cup A^{-1}$, let $\kappa'(a) = \kappa(a) - \varphi(a^*) + \varphi(a^{*-1})$; we have $\kappa'(a) = -\kappa'(a^{-1})$, hence this defines a length function. If $a \in A_x$, we have $\varphi(a^*) \leq \kappa(a)$, so $\kappa'(a) \geq 0$. If a belongs to γ, we have $\varphi(a^*) = \kappa(a)$ and $\varphi(a^{*-1}) = 0$, so $\kappa'(a) = 0$.

We claim that $\kappa' \sim \kappa$. By the flow conservation law in H^*, $\kappa' - \kappa$ is a linear combination of functions of the form z^γ, where γ^* is an (s^*, t^*)-path or a circuit in H^*; so it suffices to prove that $z^{\gamma\top}\delta = 0$ for each closed walk δ with winding number zero. But $z^{\gamma\top}\delta$ equals the number of times δ crosses γ^* from left to right minus the number of times δ crosses γ^* from right to left. This always equals zero if γ is a circuit; if γ is an (s^*, t^*)-path, this equals zero because δ has winding number zero (as in the proof of Lemma 3.1). This proves $\kappa' \sim \kappa$.

Now, let $x' = x + z^\gamma$. The length function κ' is nonnegative on the arcs of D_x and is zero on the arcs of γ, so it is nonnegative on $D_{x'}$. ∎

To conclude, recall that the k-flow x_0 and the length function κ_0 have been computed in Section 5; $\kappa_0 \sim \lambda$ is nonnegative on D_{x_0}; the integer w_0 is the winding number of x_0 and we have

$$w_0 - k < w_1 < w_0 < w_2 < w_0 + k.$$

Applying iteratively Proposition 7.3, we can find a (k, w_2)-flow x_2 and a length function $\kappa_2 \sim \lambda$ that is nonnegative on D_{x_2}; thus, x_2 is a (k, w_2)-flow of minimal cost with respect to λ, by Lemma 4.1(ii) and Proposition 7.1; if no such flow exists, we detect it during the course of the algorithm. Similarly, we can find a minimum-cost (k, w_1)-flow. This takes $O(kn \log n)$ time. By Propositions 6.1, 2.1, and 3.2, the cheapest of these two flows corresponds to the solution. This concludes the proof of Theorem 1.1.

Conclusion

We have given an algorithm to compute minimum-length vertex-disjoint paths connecting prescribed pairs (s_i, t_i) of terminals in a planar graph, where the s_i and the t_i are incident, respectively, with given faces s and t. The running time is $O(kn \log n)$, where k is the number of pairs of terminals and n is the complexity of the graph.

We note that the techniques developed above allow to solve the same problem, but fixing, in addition, the winding number of the set of paths (or, equivalently, the homotopy classes of the paths in the annulus $\mathbf{R}^2 \setminus \{s \cup t\}$). This can be done by computing a minimum-cost flow in the directed graph D and by rotating the flow until achieving the correct winding number. Since the absolute value of the winding number of a flow is at most n, the complexity of the algorithm is $O(n^2 \log n)$.

Finally, the result of this paper suggests some open questions. How hard is it to solve the minimum-length vertex-disjoint paths in case (a) of the introduction, namely, if all terminals lie on the outer face (not necessarily in the order $s_1, \ldots, s_k, t_k, \ldots, t_1$)? And in the case where all the terminals lie on two faces, but a path may have its two endpoints on the same face? The problem extends to vertex-disjoint *trees* whose leaves are fixed on two faces of the graph (such trees, not necessarily of minimal length, can be computed efficiently [SAN90]). Also, does our problem remain polynomial-time solvable if each of the terminals has to be incident with one of p prescribed faces of the graph, if p is fixed? What about the same problem for a graph embedded on a surface of fixed genus?

Acknowledgements

We would like to thank Dion Gijswijt and Günter Rote for stimulating discussions.

References

[BK06] G. Borradaile and P. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 524–533, 2006.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, pages 269–271, 1959.

[FT87] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved netweok optimization algorithms. *Journal of the Association for Computing Machinery*, 34:596–615, 1987.

[HKRS97] M. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1, part 1):3–23, 1997.

[KvL84] M. R. Kramer and J. van Leeuwen. The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits. In F. P. Preparata, editor, *VLSI-Theory*, volume 2 of *Advances in Computing Research*, pages 129–146. JAI Press, Greenwich, Connecticut, 1984.

[Rag86] P. Raghavan. *Randomized rounding and discrete ham-sandwich theorems: provably good algorithms for routing and packing problems*. PhD thesis, University of California, Berkeley, California, 1986. Report No. UCB/CSD 87/312.

[Rei83] J. H. Reif. Minimum $s-t$ cut of a planar undirected network in $O(n \log^2(n))$ time. *SIAM Journal on Computing*, 12(1):71–81, 1983.

[RLWW96] H. Ripphausen-Lipa, D. Wagner, and K. Weihe. Linear-time algorithms for disjoint two-face paths problems in planar graphs. *International Journal of Foundations of Computer Science*, 7(2):95–110, 1996.

[RS95] N. Robertson and P. D. Seymour. Graph minors. XIII: the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

[SAN90] H. Suzuki, T. Akama, and T. Nishizeki. Finding Steiner forests in planar graphs. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 444–453, 1990.

[Sch94] A. Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4):780–788, 1994.

[Sch03] A. Schrijver. *Combinatorial optimization. Polyhedra and efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, 2003.

[TSN96] J. Takahashi, H. Suzuki, and T. Nishizeki. Shortest noncrossing paths in plane graphs. *Algorithmica*, 16:339–357, 1996.

[vdHdP02] H. van der Holst and J. C. de Pina. Length-bounded disjoint paths in planar graphs. *Discrete Applied Mathematics*, 120(1–3):251–261, August 2002.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 193-204
www.stacs-conf.org

GEODESIC FRÉCHET DISTANCE INSIDE A SIMPLE POLYGON

ATLAS F. COOK IV AND CAROLA WENK

Department of Computer Science, University of Texas at San Antonio
One UTSA Circle, San Antonio, TX 78249-0667
E-mail address: {acook,carola}@cs.utsa.edu

Abstract. We unveil an alluring alternative to parametric search that applies to both the non-geodesic and geodesic Fréchet optimization problems. This randomized approach is based on a variant of red-blue intersections and is appealing due to its elegance and practical efficiency when compared to parametric search.

We present the first algorithm for the geodesic Fréchet distance between two polygonal curves A and B inside a simple bounding polygon P. The geodesic Fréchet decision problem is solved almost as fast as its non-geodesic sibling and requires $O(N^2 \log k)$ time and $O(k+N)$ space after $O(k)$ preprocessing, where N is the larger of the complexities of A and B and k is the complexity of P. The geodesic Fréchet optimization problem is solved by a randomized approach in $O(k+N^2 \log kN \log N)$ expected time and $O(k+N^2)$ space. This runtime is only a logarithmic factor larger than the standard non-geodesic Fréchet algorithm [4]. Results are also presented for the geodesic Fréchet distance in a polygonal domain with obstacles and the geodesic Hausdorff distance for sets of points or sets of line segments inside a simple polygon P.

1. Introduction

The comparison of geometric shapes is essential in various applications including computer vision, computer aided design, robotics, medical imaging, and drug design. The Fréchet distance is a similarity metric for continuous shapes such as curves or surfaces which is defined using reparametrizations of the shapes. Since it takes the continuity of the shapes into account, it is generally a more appropriate distance measure than the often used Hausdorff distance. The Fréchet distance for curves is commonly illustrated by a person walking a dog on a leash [4]. The person walks forward on one curve, and the dog walks forward on the other curve. As the person and dog move along their respective curves, a leash is maintained to keep track of the separation between them. The Fréchet distance is the length of the *shortest* leash that makes it possible for the person and dog to walk from beginning to end on their respective curves without breaking the leash. See section 2 for a formal definition of the Fréchet distance.

1998 ACM Subject Classification: Computational Geometry.
Key words and phrases: Fréchet Distance, Geodesic, Parametric Search, Simple Polygon.
The full version of this paper is available as a technical report [10].
This work has been supported by the National Science Foundation grant NSF CAREER CCF-0643597.

Most previous work assumes an obstacle-free environment where the leash connecting the person to the dog has its length defined by an L_p metric. In [4] the Fréchet distance between polygonal curves A and B is computed in arbitrary dimensions for obstacle-free environments in $O(N^2 \log N)$ time, where N is the larger of the complexities of A and B. Rote [23] computes the Fréchet distance between piecewise smooth curves. Buchin et al. [7] show how to compute the Fréchet distance between two simple polygons. Fréchet distance has also been used successfully in the practical realm of map matching [26]. All these works assume a leash length that is defined by an L_p metric.

This paper's contribution is to measure the leash length by its geodesic distance inside a simple polygon P (instead of by its L_p distance). To our knowledge, there are only two other works that employ such a leash. One is a workshop article [18] that computes the Fréchet distance for polygonal curves A and B on the surface of a convex polyhedron in $O(N^3k^4 \log(kN))$ time. The other paper [12] applies the Fréchet distance to morphing by considering the polygonal curves A and B to be obstacles that the leash must go around. Their method works in $O(N^2 \log^2 N)$ time but only applies when A and B both lie on the boundary of a simple polygon. Our work can handle both this case and more general cases. We consider a simple polygon P to be the only obstacle and the curves, which may intersect each other or self-intersect, both lie inside P.

A core insight of this paper is that the free space in a geodesic cell (see section 2) is x-monotone, y-monotone, and connected. We show how to quickly compute a cell boundary and how to propagate reachability through a cell in constant time. This is sufficient to solve the geodesic Fréchet decision problem. To solve the geodesic Fréchet optimization problem, we replace the standard parametric search approach by a novel and asymptotically faster (in the expected case) randomized algorithm that is based on red-blue intersection counting. We show that the geodesic Fréchet distance between two polygonal curves inside a simple bounding polygon can be computed in $O(k + N^2 \log kN \log N)$ expected time and $O(k + N^3 \log kN)$ worst-case time, where N is the larger of the complexities of A and B and k is the complexity of the simple polygon. The expected runtime is almost a quadratic factor in k faster than the straightforward approach, similar to [12], of partitioning each cell into $O(k^2)$ subcells. Briefly, these subcells are simple combinatorial regions based on *pairs* of hourglass intervals. It is notable that the randomized algorithm also applies to the non-geodesic Fréchet distance in arbitrary dimensions. We also present algorithms to compute the geodesic Fréchet distance in a polygonal domain with obstacles and the geodesic Hausdorff distance for sets of points or sets of line segments inside a simple polygon.

2. Preliminaries

Let k be the complexity of a simple polygon P that contains polygonal curves A and B in its interior. In general, a *geodesic* is a path that avoids all obstacles and cannot be shortened by slight perturbations [20]. However, a geodesic inside a simple polygon is simply a unique shortest path between two points. Let $\pi(a, b)$ denote the geodesic inside P between points a and b. The *geodesic distance* $d(a, b)$ is the length of a shortest path between a and b that avoids all obstacles, where length is measured by L_2 distance.

Let $\downarrow$, $\uparrow$, and $\downarrow\uparrow$ denote decreasing, increasing, and decreasing then increasing functions, respectively. For example, "H is $\downarrow\uparrow$-bitonic" means that H is a function that decreases monotonically then increases monotonically. A *bitonic* function has at most one change in monotonicity.

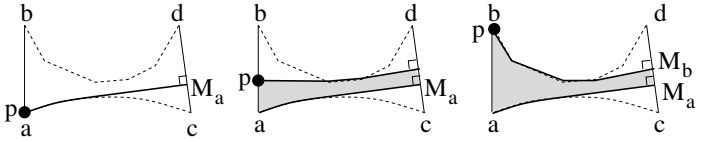


Figure 1: Shortest paths in the hourglass $\mathcal{H}_{\overline{ab},\overline{cd}}$ define $H_{\overline{ab},\ \overline{cd}}$.

The Fréchet distance for two curves $A, B : [0,1] \to \mathbb{R}^l$ is defined as

$$\delta_F(A,B) = \inf_{f,g:[0,1]\to[0,1]} \sup_{t\in[0,1]} d'(\ A(f(t)), B(g(t))\ )$$

where f and g range over continuous non-decreasing reparametrizations and d' is a distance metric for points, usually the L_2 distance, and in our setting the geodesic distance. For a given $\varepsilon > 0$ the *free space* is defined as $FS_\varepsilon(A,B) = \{(s,t) \mid d'(A(s), B(t)) \le \varepsilon\} \subseteq [0,1]^2$. A free space cell $C \subseteq [0,1]^2$ is the parameter space defined by two line segments $\overline{ab} \in A$ and $\overline{cd} \in B$, and the free space inside the cell is $FS_\varepsilon(\overline{ab}, \overline{cd}) = FS_\varepsilon(A,B) \cap C$.

The decision problem to check whether the Fréchet distance is at most a given $\varepsilon > 0$ is solved by Alt and Godau [4] using a *free space diagram* which consists of all free space cells for all pairs of line segments of A and B. Their dynamic programming algorithm checks for the existence of a monotone path in the free space from $(0,0)$ to $(1,1)$ by propagating *reachability information* cell by cell through the free space.

2.1. Funnels and Hourglasses

Geodesics in a free space cell C can be described by either the funnel or hourglass structure of [14]. A funnel describes all shortest paths between a point and a line segment, so it represents a horizontal (or vertical) line segment in C. An hourglass describes all shortest paths between two line segments and represents all distances in C.

The *funnel* $\mathcal{F}_{p,\overline{cd}}$ describes all shortest paths between an apex point p and a line segment $\overline{cd}$. The boundary of $\mathcal{F}_{p,\overline{cd}}$ is the union of the line segment $\overline{cd}$ and the shortest path chains $\pi(p,c)$ and $\pi(p,d)$. The *hourglass* $\mathcal{H}_{\overline{ab},\overline{cd}}$ describes all shortest paths between two line segments $\overline{ab}$ and $\overline{cd}$. The boundary of $\mathcal{H}_{\overline{ab},\overline{cd}}$ is composed of the two line segments $\overline{ab}$, $\overline{cd}$ and at most four shortest path chains involving a, b, c, and d. See Figure 1. Funnel and hourglass boundaries have $O(k)$ complexity because shortest paths inside a simple polygon P are acyclic, polygonal, and only have corners at vertices of P [15].

Any horizontal or vertical line segment in a geodesic free space cell is associated with a funnel's distance function $F_{p,\ \overline{cd}} : [c,d] \to \mathbb{R}$ with $F_{p,\ \overline{cd}}(q) = d(p,q)$. The below three results are generalizations of Euclidean properties and are omitted. See [10] for details.

Lemma 2.1. *$F_{p,\ \overline{cd}}$ is $\downarrow\uparrow$-bitonic.*

Corollary 2.2. *Any horizontal (or vertical) line segment in a free space cell has at most one connected set of free space values.*

Consider the hourglass $\mathcal{H}_{\overline{ab},\ \overline{cd}}$ in Figure 1. Let the *shortest* distance from a to any point on $\overline{cd}$ occur at $M_a \in \overline{cd}$. Define M_b similarly. As p varies from a to b, the *minimum* distance from p to $\overline{cd}$ traces out a function $H_{\overline{ab},\ \overline{cd}} : [a,b] \to \mathbb{R}$ with $H_{\overline{ab},\ \overline{cd}}(p) = \min_{q\in[c,d]} d(p,q)$.

Lemma 2.3. *$H_{\overline{ab},\ \overline{cd}}$ is $\downarrow\uparrow$-bitonic.*

3. Geodesic Cell Properties

Consider a geodesic free space cell C for polygonal curves A and B inside a simple polygon. Let $\overline{ab} \in A$ and $\overline{cd} \in B$ be the two line segments defining C.

Lemma 3.1. *For any ε, cell C contains at most one free space region R, and R is x-monotone, y-monotone, and connected.*

Proof. The monotonicity of R follows from Corollary 2.2. For connectedness, choose any two free space points $(p_1, q_1), (p_2, q_2)$, and construct a path connecting them in the free space as follows: move vertically from (p_1, q_1) to the minimum point on its vertical. Do the same for (p_2, q_2). By Lemma 2.1, this movement causes the distance to decrease monotonically. By Lemma 2.3, any two minimum points are connected by a $\downarrow\uparrow$-bitonic distance function $H_{\overline{ab},\ \overline{cd}}$ (cf. section 2.1), but as the starting points are in the free space – and therefore have distance at most ε – all points on this constructed path lie in the free space. ∎

Given C's boundaries, it is possible to propagate reachability information (see section 2) through C in constant time. This follows from the monotonicity and connectedness of the free space in C and is useful for solving the geodesic decision problem.

4. Red-Blue Intersections

This section shows how to efficiently count and report a certain type of red-blue intersections in the plane. This problem is interesting both from theoretical and applied stances and will prove useful in section 5.3 for the Fréchet optimization problem.

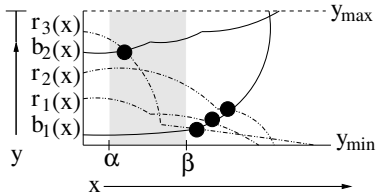
Let R be a set of m "red" curves in the plane such that every red curve is continuous, x-monotone, and monotone *decreasing*. Let B be a set of n "blue" curves in the plane where each blue curve is continuous, x-monotone, and monotone *increasing*. Assume that the curves are defined in the slab $[\alpha, \beta] \times \mathbb{R}$, and let $I(k)$ be the time to find the at most one intersection of any red and blue curve.[1]

Theorem 4.1. *The number of red-blue intersections between R and B in the slab $[\alpha, \beta] \times \mathbb{R}$ can be* counted *in $O(N \log N)$ total time, where $N = \max(m, n)$. These intersections can be* reported *in $O(N \log N + K \cdot I(k))$ total time, where K is the total number of intersections reported. After $O(N \log N)$ preprocessing time, a* random *red-blue intersection in $[\alpha, \beta] \times \mathbb{R}$ can be returned in $O(\log N + I(k))$ time, and the red curve involved in the most red-blue intersections can be returned in $O(1)$ time. All operations require $O(N)$ space.*[2]

Proof Sketch. Figure 2 illustrates the key idea. Suppose a red curve $r_3(x)$ lies *above* a blue curve $b_2(x)$ at $x = \alpha$. If it is also true that $r_3(x)$ lies *below* $b_2(x)$ at $x = \beta$, then these monotone curves must intersect in $[\alpha, \beta] \times \mathbb{R}$. Two sorted lists L_α, L_β of curve values store how many blue curves lie below each red curve at $x = \alpha$ and $x = \beta$. Subtracting the values in L_α and L_β yields the number of actual intersections for each red curve in $[\alpha, \beta] \times \mathbb{R}$ (and

[1] There is at most one intersection due to the monotonicities of the red and blue curves.

[2] Palazzi and Snoeyink [21] also count and report red-blue intersections using a slab-based approach. However, their work is for line segments instead of curves, and they require that all red segments are disjoint and all blue segments are disjoint. We have no such disjointness requirement.



Index	L_α		L_β	
5	$r_3(\alpha)$	2	$b_2(\beta)$	
4	$b_2(\alpha)$		$r_2(\beta)$	1
3	$r_2(\alpha)$	1	$r_1(\beta)$	1
2	$r_1(\alpha)$	1	$r_3(\beta)$	1
1	$b_1(\alpha)$		$b_1(\beta)$	

Figure 2: $r_3(x)$ lies above *two* blue curves at $x = \alpha$ but only lies above *one* blue curve at $x = \beta$. Subtraction reveals that $r_3(x)$ has one intersection in the slab $[\alpha, \beta] \times \mathbb{R}$.

also reveals the red curve that is involved in the most intersections). Intersection *counting* simply sums up these values. Intersection *reporting* builds a balanced tree from L_α and L_β.

To find a *random* red-blue intersection in $[\alpha, \beta] \times \mathbb{R}$, precompute the number κ of red-blue intersections in $[\alpha, \beta] \times \mathbb{R}$. Pick a random integer between 1 and κ and use the number of intersections stored for each red curve to locate the particular red curve $r_i(x)$ that is involved in the randomly selected intersection. By searching a persistent version of the reporting structure [24], $r_i(x)$'s jth red-blue intersection can be returned in $O(\log N + I(k))$ query time after $O(N \log N)$ preprocessing time. ∎

5. Geodesic Fréchet Algorithm

5.1. Computing One Cell's Boundaries in $O(\log k)$ Time

A boundary of a free space cell is a horizontal (or vertical) line segment. This boundary can be associated with a funnel $\mathcal{F}_{p,\overline{cd}}$ that has a $\downarrow\uparrow$-bitonic distance function $F_{p,\,\overline{cd}}$ (cf. Lemma 2.1). Given $\varepsilon \geq 0$, computing the free space on a cell boundary requires finding the (at most two) values t_1, t_2 such that $F_{p,\,\overline{cd}}(t_1) = F_{p,\,\overline{cd}}(t_2) = \varepsilon$ (see Figure 3).

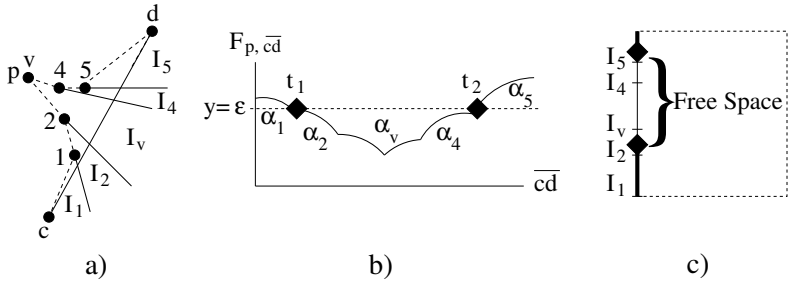
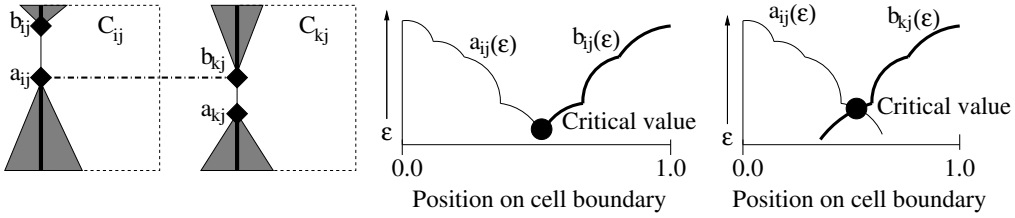


Figure 3: a & b) A funnel $\mathcal{F}_{p,\,\overline{cd}}$ is associated with a cell boundary and has a bitonic distance function $F_{p,\,\overline{cd}}$. c) The (at most two) values t_1, t_2 such that $F_{p,\,\overline{cd}}(t_1) = F_{p,\,\overline{cd}}(t_2) = \varepsilon$ define the free space on a cell boundary.

Lemma 5.1. *Both the minimum value of $F_{p,\,\overline{cd}}$ and the (at most two) values t_1, t_2 such that $F_{p,\,\overline{cd}}(t_1) = F_{p,\,\overline{cd}}(t_2) = \varepsilon$ can be found for any $\varepsilon \geq 0$ in $O(\log k)$ time (after preprocessing).*

Proof Sketch. After $O(k)$ shortest path preprocessing [13, 16], a binary search is performed on the $O(k)$ arcs of $F_{p,\,\overline{cd}}$ in $O(\log k)$ time. See our full paper [10] for details. ∎



a) Free Space Diagram

b) Distance function with a type (b) critical value

c) Distance function with a type (c) critical value

Figure 4: Critical values of the Fréchet distance

Corollary 5.2. *The free space on all four boundaries of a free space cell can be found in* $O(\log k)$ *time by computing* t_1 *and* t_2 *for each boundary.*

5.2. Geodesic Fréchet Decision Problem

Theorem 5.3. *After preprocessing a simple polygon* P *for shortest path queries in* $O(k)$ *time* [13]*, the geodesic Fréchet decision problem for polygonal curves* A *and* B *inside* P *can be solved for any* $\varepsilon \geq 0$ *in* $O(N^2 \log k)$ *time and* $O(k+N)$ *space.*

Proof. Following the standard dynamic programming approach of [4], compute all cell boundaries in $O(N^2 \log k)$ time (cf. Corollary 5.2), and propagate reachability information through all cells in $O(N^2)$ time. $O(k)$ space is needed for the preprocessing structures of [13], and only $O(N)$ space is needed for dynamic programming if two rows of the free space diagram are stored at a time. ∎

5.3. Geodesic Fréchet Optimization Problem

Let ε^* be the minimum value of ε such that the Fréchet decision problem returns true. That is, ε^* equals the Fréchet distance $\delta_F(A, B)$. Parametric search is a technique commonly used to find ε^* (see [3, 4, 9, 25]).[3] The typical approach to find ε^* is to sort all the cell boundary functions based on the unknown parameter ε^*. The comparisons performed during the sort guarantee that the result of the decision problem is known for all "critical values" [4] that could potentially define ε^*. Traditionally, such a sort operates on cell boundaries of constant complexity. The geodesic case is different because each cell boundary has $O(k)$ complexity. As a result, a straightforward parametric search based on sorting these values would require $O(kN^2 \log kN)$ time even when using Cole's [9] optimization.[4]

We present a randomized algorithm with expected runtime $O(k + N^2 \log kN \log N)$ and worst-case runtime $O(k + N^3 \log kN)$. This algorithm is an order of magnitude faster than parametric search in the expected case.

Each cell boundary has at most one free space interval (cf. Lemma 2.1). The upper boundary of this interval is a function $b_{ij}(\varepsilon)$, and the lower boundary of this interval is a function $a_{ij}(\varepsilon)$. See Figure 4a. The seminal work of Alt and Godau [4] defines three types

[3] An easier to implement alternative to parametric search is to run the decision problem once for every bit of accuracy that is desired. This approach runs in $O(BN^2 \log k)$ time and $O(k+N)$ space, where B is the desired number of bits of accuracy [25].

[4] A variation of the general sorting problem called the "nuts and bolts" problem (see [17]) is tantalizingly close to an acceptable $O(N^2 \log N)$ sort but does not apply to our setting.

of critical values that are useful for computing the exact geodesic Fréchet distance. There are exactly two type (a) critical values associated with distances between the starting points of A and B and the ending points of A and B. Type (b) critical values occur $O(N^2)$ times when $a_{ij}(\varepsilon) = b_{ij}(\varepsilon)$. See Figure 4b. Type (a) and (b) critical values occur $O(N^2)$ times and are easily handled in $O(N^2 \log k \log N)$ time. This process involves computing values in $O(N^2 \log k)$ time, sorting in $O(N^2 \log N)$ time, and running the decision problem in binary search fashion $O(\log N)$ times. Resolving the type (a) and (b) critical values as a first step will simplify the randomized algorithm for the type (c) critical values.

Alt and Godau [4] show that type (c) critical values occur when the position of $a_{ij}(\varepsilon)$ in cell C_{ij} equals the position of $b_{kj}(\varepsilon)$ in cell C_{kj} in the free space diagram. See Figure 4a. As ε increases, by Lemma 2.1, $a_{ij}(\varepsilon)$ is $\downarrow$-monotone on the cell boundary and $b_{ij}(\varepsilon)$ is $\uparrow$-monotone (see Figure 4b). As illustrated in Figure 4c, $a_{ij}(\varepsilon)$ and $b_{kj}(\varepsilon)$ intersect at most once. This follows from the monotonicities of $a_{ij}(\varepsilon)$ and $b_{kj}(\varepsilon)$. Hence, there are $O(N^2)$ intersections of $a_{ij}(\varepsilon)$ and $b_{kj}(\varepsilon)$ in row j and a total of $O(N^3)$ type (c) critical values over all rows. There are also $O(N^2)$ intersections of $a_{ij}(\varepsilon)$ and $b_{ik}(\varepsilon)$ in *column* i and a total of $O(N^3)$ additional type (c) critical values over all columns.

Lemma 5.4. *The intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ can be found for any $\varepsilon \geq 0$ in $O(\log k)$ time after preprocessing.*

Proof Sketch. Build binary search trees for $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ and perform a binary search. See our full paper [10] for details. ∎

Theorem 4.1 requires that all $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ are defined in the slab $[\alpha, \beta] \times \mathbb{R}$ that contains ε^*. Precomputing the type (a) and type (b) critical values of [4] shrinks the slab such that no *left* endpoint of any relevant $a_{ij}(\varepsilon)$, $b_{kl}(\varepsilon)$ appears in $[\alpha, \beta] \times \mathbb{R}$ when processing the type (c) critical values. In addition, $a_{ij}(\varepsilon)$, $b_{kl}(\varepsilon)$ can be extended horizontally so that no *right* endpoint appears in $[\alpha, \beta] \times \mathbb{R}$. These changes do not affect the asymptotic number of intersections and allow Theorem 4.1 to count and report type (c) critical values in $[\alpha, \beta] \times \mathbb{R}$.

The below randomized algorithm solves the geodesic Fréchet optimization problem in $O(k + N^2 \log kN \log N)$ expected time. This is faster than the standard parametric search approach which requires $O(kN^2 \log kN)$ time.

Randomized Optimization Algorithm

(1) Precompute and sort all type (a) and type (b) critical values in $O(N^2 \log kN)$ time (cf. Lemma 5.1). Run the decision problem $O(\log N)$ times to resolve these values and shrink the potential slab for ε^* down to $[\alpha, \beta] \times \mathbb{R}$ in $O(N^2 \log k \log N)$ time.
(2) Count the number κ_j of type (c) critical values for each row j in the slab $[\alpha, \beta] \times \mathbb{R}$ using Theorem 4.1. Let C_j be the resulting counting data structure for row j.
(3) To achieve a fast *expected* runtime, pick a random intersection ϑ_j for each row using C_j.[5] See Theorem 4.1.
(4) To achieve a fast *worst-case* runtime, use C_j to find the $a_{Mj}(\varepsilon)$ curve in each row that has the most intersections (see Theorem 4.1). Add all intersections in $[\alpha, \beta] \times \mathbb{R}$ that involve $a_{Mj}(\varepsilon)$ to a global pool $\mathcal{P}$ of unresolved critical values[6] and delete $a_{Mj}(\varepsilon)$ from any future consideration.

[5]Picking a critical value at random is related to the distance selection problem [6] and is mentioned in [2], but to our knowledge, this alternative to parametric search has never been applied to the Fréchet distance.

[6]The idea of a global pool is similar to Cole's optimization for parametric search [9].

(5) Find the median Ξ of the values in $\mathcal{P}$ in $O(N^2)$ time using the standard median algorithm mentioned in [17]. Also find the median Ψ of the $O(N)$ randomly selected ϑ_j in $O(N)$ time using a *weighted* median algorithm based on the number of critical values κ_j for each row j.
(6) Run the decision problem twice: once on Ξ and once on Ψ. This shrinks the search slab $[\alpha, \beta] \times \mathbb{R}$ and *at least* halves the size of $\mathcal{P}$. Repeat steps 2 through 6 until all *row*-based type (c) critical values have been resolved.
(7) Resolve all *column*-based type (c) critical values in the same spirit as steps 2 through 6 and return the smallest critical value that satisfied the decision problem as the value of the geodesic Fréchet distance.

Theorem 5.5. *The exact* geodesic *Fréchet distance between two polygonal curves A and B inside a simple bounding polygon P can be computed in $O(k + N^2 \log kN \log N)$ expected time and $O(k + N^3 \log kN)$ worst-case time, where N is the larger of the complexities of A and B and k is the complexity of P. $O(k + N^2)$ space is required.*

Proof. Preprocess P once for shortest path queries in $O(k)$ time [13]. In the expected case, each execution of the decision problem will eliminate a constant fraction of the remaining type (c) critical values due to the proof of Quicksort's expected runtime and the median of medians approach for Ψ. Consequently, the expected number of iterations of the algorithm is $O(\log N^3) = O(\log N)$.

In the worst-case, each of the $O(N)$ $a_{ij}(\varepsilon)$ in a row will be picked as $a_{Mj}(\varepsilon)$. Therefore, each row can require at most $O(N)$ iterations. Since *all* rows are processed each iteration, the entire algorithm requires at most $O(N)$ iterations for *row*-based critical values. By a similar argument, *column*-based critical values also require at most $O(N)$ iterations.

The size of the pool $\mathcal{P}$ is expressed by the inequality $S(x) \leq \frac{S(x-1)+O(N^2)}{2}$, where x is the current step number, and $S(0) = 0$. Intuitively, each step adds $O(N^2)$ values to $\mathcal{P}$ and then at least half of the values in $\mathcal{P}$ are always resolved using the median Ξ. It is not difficult to show that $S(x) \in O(N^2)$ for any step number x.

Each iteration of the algorithm requires intersection counting and intersection calculations for $O(N)$ rows (or columns) at a cost of $O(N^2 \log kN)$ time. In addition, the global pool $\mathcal{P}$ has its median calculated in $O(N^2)$ time, and the decision problem is executed in $O(N^2 \log k)$ time. Consequently, the expected runtime is $O(k + N^2 \log kN \log N)$ and the worst-case runtime is $O(k+N^3 \log kN)$ including $O(k)$ preprocessing time [13] for geodesics. The preprocessing structures use $O(k)$ space that must remain allocated throughout the algorithm, and the pool $\mathcal{P}$ uses $O(N^2)$ additional space. ∎

Although the exact non-geodesic Fréchet distance is normally found in $O(N^2 \log N)$ time using parametric search (see [4]), parametric search is often regarded as impractical because it is difficult to implement[7] and involves enormous constant factors [9]. To the best of our knowledge, the randomized algorithm in section 5.3 provides the first practical alternative to parametric search for solving the exact non-geodesic Fréchet optimization problem in $\mathbb{R}^l$.

Theorem 5.6. *The exact* non-geodesic *Fréchet distance between two polygonal curves A and B in $\mathbb{R}^l$ can be computed in $O(N^2 \log^2 N)$ expected time, where N is the larger of the complexities of A and B. $O(N^2)$ space is required.*

[7]Quicksort-based parametric search has been implemented by van Oostrum and Veltkamp [25] using a complex framework.

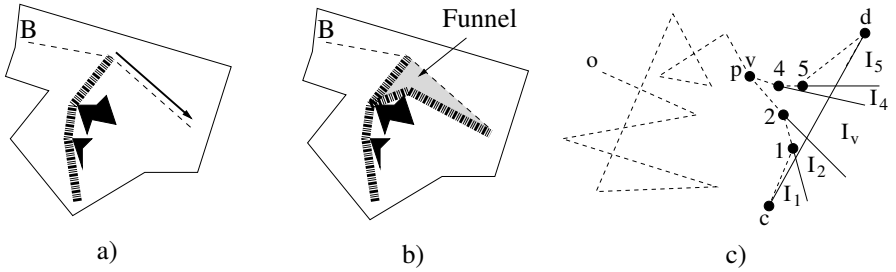


Figure 5: a) A funnel for a δ_C-cell can be found by extending a cell's initial leash along one segment to create a path sketch and then b) snapping this sketch into a homotopic shortest path. c) A funnel $\mathcal{F}_{o,\,\overline{cd}}$ has $O(kN)$ complexity, but the distance function $F_{o,\,\overline{cd}}$ has only $O(k)$ complexity because $d(o,p)$ is a constant.

Proof. The argument is very similar to the proof of Theorem 5.5. The main difference is that non-geodesic distances can be computed in $O(1)$ time (instead of $O(\log k)$ time). ∎

6. Geodesic Fréchet Distance in a Polygonal Domain with Obstacles

Consider the real-life situation of a person walking a dog in a park. If the person and dog walk on opposite sides of a group of trees, then the leash must go around the trees. More formally, suppose the two polygonal curves A and B lie in a planar polygonal domain $\mathcal{D}$ [19] of complexity k. The leash is required to change continuously, i.e., it must stay inside $\mathcal{D}$ and may not pass through or jump over an obstacle. It may, however, cross itself. Let δ_C be the geodesic Fréchet distance for this scenario when the leash length is measured geodesically.[8]

Due to the continuity of the leash's motion, the free space inside a geodesic cell is represented by an hourglass – just as it was for the geodesic Fréchet distance inside a simple polygon. Hence, free space in a cell is x-monotone, y-monotone, and connected (cf. Lemma 3.1), and reachability information can be propagated through a cell in constant time.

The main task in computing δ_C is to construct all cell boundaries. Once the cell boundaries are known, the decision and optimization problems can be solved by the algorithms for the geodesic Fréchet distance inside a simple polygon (cf. Theorems 5.3 and 5.5). We use Hershberger and Snoeyink's homotopic shortest paths algorithm [16] to incrementally construct all cell boundary funnels needed to compute δ_C. To use the homotopic algorithm, the polygonal domain $\mathcal{D}$ should be triangulated in $O(k \log k)$ time [19], and all obstacles should be replaced by their vertices. A shortest path map [19] can find an initial geodesic leash L_I between the start points of the polygonal curves A and B in $O(k \log k)$ time.

Lemma 6.1. *Given the initial leash for the bottom-left corner of a δ_C-cell $\mathcal{C}$, all four funnel boundaries of $\mathcal{C}$ and the initial leashes for cells adjacent to $\mathcal{C}$ can be computed in $O(k)$ time.*

Proof. The funnels representing cell boundaries are constructed *incrementally*. The idea is to extend the initial leash into a homotopic "sketch" that describes how the shortest path should wind through the obstacles and then to "snap" this sketch into a shortest path (see Figures 5a and 5b).

[8] We recently learned that this topic has been independently explored in [8].

Homotopic shortest paths have increased complexity over normal shortest paths because they can loop around obstacles. For example, if the person walks in a triangular path around all the obstacles, then the leash follows a homotopic shortest path that can have $O(k)$ complexity in a single cycle around the obstacles. By repeatedly winding around the obstacles $O(N)$ times, a path achieves $O(kN)$ complexity.

To avoid spending $O(kN)$ time per cell, we extend a previous homotopic shortest path into a sketch by appending a single line segment to the previous path (see Figure 5a). Adding this single segment can unwind at most one loop over a subset of obstacles, so only the most recent $O(k)$ vertices of the sketch will need to be updated when the sketch is snapped into the true homotopic shortest path. A turning angle is used to identify these $O(k)$ vertices by backtracking on the sketch until the angle is at least 2π different from the final angle.

Putting all this together, a boundary for a free space cell can be computed in $O(k)$ time by starting with an initial leash L_I of $O(kN)$ complexity, constructing a homotopic sketch by appending a single segment to L_I, backtracking with a turning angle to find $O(k)$ vertices that are eligible to be changed, and finally "snapping" these $O(k)$ vertices to the true homotopic shortest path using Hershberger and Snoeyink's algorithm [16]. The result is a funnel that describes one cell boundary.

By extending L_I in four combinatorially distinct ways, all four cell boundaries can be defined. Specifically, we can extend L_I along the current $\overline{ab} \in A$ segment to form the first funnel or along the $\overline{cd} \in B$ segment to form the second funnel. The third funnel is created by extending L_I along $\overline{ab} \in A$ and then $\overline{cd} \in B$. The fourth funnel is created by extending L_I along $\overline{cd} \in B$ and then $\overline{ab} \in A$. These cell boundaries conveniently define the initial leash for cells that are adjacent to $\mathcal{C}$. ∎

Theorem 6.2. *The δ_C decision problem can be solved in $O(kN^2)$ time and $O(k+N)$ space.*

Proof. Each cell boundary is a funnel $\mathcal{F}_{o,\,\overline{cd}}$ with $O(kN)$ complexity [11]. However, this high complexity is a result of looping over obstacles, and most of these points do not affect the funnel's distance function $F_{o,\,\overline{cd}}$. As illustrated in Figure 5c, $F_{o,\,\overline{cd}}$ has only $O(k)$ complexity because only vertices $\pi(p,c) \cup \pi(p,d)$ contribute arcs to $F_{o,\,\overline{cd}}$.

Construct all cell boundary funnels in $O(kN^2)$ time (cf. Lemma 6.1), intersect each funnel's distance function with $y = \varepsilon$ in $O(N^2 \log k)$ time, and propagate reachability information in $O(N^2)$ time. Only $O(k+N)$ space is needed for dynamic programming when storing only two rows at a time. ∎

Theorem 6.3. *The δ_C optimization problem can be solved in $O(kN^2 + N^2 \log kN \log N)$ expected time and $O(kN^2)$ space.*[9]

Proof. The δ_C optimization problem can be solved using red-blue intersections. $O(\log N)$ steps are performed in the expected case by Theorem 5.5. Each step has to perform intersection counting in $O(N^2 \log kN)$ time and solve the decision problem. If the funnels are precomputed in $O(kN^2)$ time and space, then the decision problem can be solved in $O(N^2 \log k)$ time. Hence, after $O(kN^2)$ time and space preprocessing, δ_C can be found in $O(\log N)$ expected steps where each step takes $O(N^2 \log kN)$ time. ∎

[9]If space is at a premium, the algorithm can also run with $O(k + N^2)$ space and $O(kN^2 \log N + N^2 \log kN \log N)$ expected time by recomputing the funnels each time the decision problem is computed. Note that $O(N^2)$ storage is required for the red-blue intersections algorithm (cf. Theorem 5.5).

7. Geodesic Hausdorff Distance

Hausdorff distance is a similarity metric commonly used to compare sets of points or sets of line segments. The *directed* geodesic Hausdorff distance can be formally defined as $\tilde{\delta}_H(A,B) = \sup_{a\in A}\inf_{b\in B} d(a,b)$, where A and B are sets and $d(a,b)$ is the geodesic distance between a and b (see [4, 5]). The *undirected* geodesic Hausdorff distance is the larger of the two directed distances: $\delta_H(A,B) = \max(\tilde{\delta}_H(A,B),\ \tilde{\delta}_H(B,A))$.

Theorem 7.1. *$\delta_H(A,B)$ for point sets A, B inside a simple polygon P can be computed in $O((k+N)\log(k+N))$ time and $O(k+N)$ space, where N is the larger of the complexities of A and B and k is the complexity of P. If A and B are sets of line segments, $\delta_H(A,B)$ can be computed in $O(kN^2\alpha(kN)\log kN)$ time and $O(kN\alpha(kN)\log kN)$ space.*

Proof Sketch. A geodesic Voronoi diagram [22] finds nearest neighbors when A and B are point sets. When A and B are sets of line segments, all nearest neighbors for a line segment can be found by computing a lower envelope [1] of $O(N)$ hourglass distance functions. The largest nearest neighbor distance over all line segments is $\delta_H(A,B)$. ∎

8. Conclusion

To compute the geodesic Fréchet distance between two polygonal curves inside a simple polygon, we have proven that the free space inside a geodesic cell is x-monotone, y-monotone, and connected. By extending the shortest path algorithms of [13, 16], the boundaries of a single free space cell can be computed in logarithmic time, and this leads to an efficient algorithm for the geodesic Fréchet decision problem.

A randomized algorithm based on red-blue intersections solves the geodesic Fréchet optimization problem in lieu of the standard parametric search approach. The randomized algorithm is also a practical alternative to parametric search for the non-geodesic Fréchet distance in arbitrary dimensions.

We can compute the geodesic Fréchet distance between two polygonal curves A and B inside a simple bounding polygon P in $O(k+N^2\log kN\log N)$ expected time, where N is the larger of the complexities of A and B and k is the complexity of P. In the expected case, the randomized optimization algorithm is an order of magnitude faster than a straightforward parametric search that uses Cole's [9] optimization to sort $O(kN^2)$ values.

The geodesic Fréchet distance in a polygonal domain with obstacles enforces a homotopy on the leash. It can be computed in the same manner as the geodesic Fréchet distance inside a simple polygon after computing cell boundary funnels using Hershberger and Snoeyink's homotopic shortest paths algorithm [16]. Future work could attempt to compute these funnels in $O(\log k)$ time instead of $O(k)$ time. The geodesic Hausdorff distance for point sets inside a simple polygon can be computed using geodesic Voronoi diagrams. The geodesic Hausdorff distance for line segments can be computed using lower envelopes; future work could speed up this algorithm by developing a geodesic Voronoi diagram for line segments.

References

[1] P. K. Agarwal and M. Sharir. Davenport–Schinzel sequences and their geometric applications. Technical Report Technical report DUKE–TR–1995–21, 1995.

[2] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30(4):412–458, 1998.
[3] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. volume 17, pages 292–318, Duluth, MN, USA, 1994. Academic Press, Inc.
[4] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995.
[5] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.
[6] S. Bespamyatnikh and M. Segal. Selecting distances in arrangements of hyperplanes spanned by points. volume 2, pages 333–345, September 2004.
[7] K. Buchin, M. Buchin, and C. Wenk. Computing the Fréchet distance between simple polygons in polynomial time. *SoCG: 22nd Symposium on Computational Geometry*, pages 80–87, 2006.
[8] E. W. Chambers, É. C. de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Walking your dog in the woods in polynomial time. *17th Fall Workshop on Computational Geometry*, 2007.
[9] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.
[10] A. F. Cook IV and C. Wenk. Geodesic Fréchet and Hausdorff distance inside a simple polygon. Technical Report CS-TR-2007-004, University of Texas at San Antonio, August 2007.
[11] C. A. Duncan, A. Efrat, S. G. Kobourov, and C. Wenk. Drawing with fat edges. *Int. J. Found. Comput. Sci.*, 17(5):1143–1164, 2006.
[12] A. Efrat, L. J. Guibas, S. Har-Peled, J. S. B. Mitchell, and T. M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete & Computational Geometry*, 28(4):535–569, 2002.
[13] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989.
[14] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. pages 1–13, 1986.
[15] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.
[16] J. Hershberger. A new data structure for shortest path queries in a simple polygon. *Inf. Process. Lett.*, 38(5):231–235, 1991.
[17] J. Komlós, Y. Ma, and E. Szemerédi. Matching nuts and bolts in O(n log n) time. *SODA: 7th ACM-SIAM Symposium on Discrete Algorithms*, pages 232–241, 1996.
[18] A. Maheshwari and J. Yi. On computing Fréchet distance of two paths on a convex polyhedron. *EWCG 2005*, pages 41–4, 2005.
[19] J. S. B. Mitchell. Geometric shortest paths and network optimization. *Handbook of Computational Geometry*, 1998.
[20] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, 1987.
[21] L. Palazzi and J. Snoeyink. Counting and reporting red/blue segment intersections. *CVGIP: Graph. Models Image Process.*, 56(4):304–310, 1994.
[22] E. Papadopoulou and D. T. Lee. A new approach for the geodesic Voronoi diagram of points in a simple polygon and other restricted polygonal domains. *Algorithmica*, 20(4):319–352, 1998.
[23] G. Rote. Computing the Fréchet distance between piecewise smooth curves. Technical Report ECG-TR-241108-01, May 2005.
[24] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Commun. ACM*, 29(7):669–679, 1986.
[25] R. van Oostrum and R. C. Veltkamp. Parametric search made practical. *SoCG: 18th Symposium on Computational Geometry*, pages 1–9, 2002.
[26] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. *18th Int'l Conf. on Sci. and Statistical Database Mgmt (SSDBM)*, pages 379–388, 2006.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 205-216
www.stacs-conf.org

IMPROVED ALGORITHMS FOR THE RANGE NEXT VALUE PROBLEM AND APPLICATIONS

MAXIME CROCHEMORE [2,1], COSTAS S. ILIOPOULOS [2], MARCIN KUBICA [3], M. SOHEL RAHMAN [2], AND TOMASZ WALEŃ [3]

[1] Institut Gaspard-Monge, Université de Marne-la-Vallée, France

[2] Algorithm Design Group, Department of Computer Science
Kings College London, Strand, London WC2R 2LS, England
E-mail address: Maxime.Crochemore@kcl.ac.uk,{csi,sohel}@dcs.kcl.ac.uk
URL: http://www.dcs.kcl.ac.uk/adg

[3] Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warszawa, Poland
E-mail address: {kubica,walen}@mimuw.edu.pl

Abstract. The Range Next Value problem (Problem RNV) is a recent interesting variant of the range search problems, where the query is for the immediate next (or equal) value of a given number within a given interval of an array. Problem RNV was introduced and studied very recently by Crochemore et. al [Finding Patterns In Given Intervals, MFCS 2007]. In this paper, we present improved algorithms for Problem RNV. We also show how this problem can be used to achieve optimal query time for a number of interesting variants of the classic pattern matching problems.

1. Introduction

We study the Range Next Value (RNV) problem, which is defined as follows:

Problem 1.1. Range Next Value (Problem RNV). We are given an array $A[1..n]$, which is a permutation of $[1..n]$. We need to preprocess A to answer queries of the following form:
Query: Given an integer $\mathcal{K} \in [1..n]$, and an interval $[\ell..r], 1 \leq \ell \leq r \leq n$, the goal is to return the value $A[k]$ of the immediate higher or equal number ('next value') than $\mathcal{K}$

Key words and phrases: Algorithms, Data structures.

Part of this research work was done when the authors were visiting McMaster University, Canada to attend StringMasters @ McMaster (2007). C.S. Iliopoulos is partially supported by the EPSRC and Royal Society grants. M. Kubica and T. Waleń are supported by the grant of the Polish Ministry of Science and Higher Education N206 004 32/0806. M.S. Rahman is supported by the Commonwealth Scholarship Commission in the UK and is on leave from the Department of CSE, BUET, Dhaka-1000, Bangladesh.

from $A[\ell..r]$ if there exists one. More formally, we need to return such $A[k]$, that $A[k] = \min\{A[q] \mid A[q] \geq \mathcal{K} \text{ and } \ell \leq q \leq r\}$. If there is no such k, then we return -1.

We use $RNV_A([\ell..r], \mathcal{K})$ to denote the range next value query on array $A[\ell..r]$ for the value $\mathcal{K}$. Problem RNV was introduced, very recently, in [4], to solve an interesting variant of the classic pattern matching problem, namely Pattern Matching in a Query Interval (Problem PMQI) [8]. In problem PMQI, we are given a text, which we can preprocess for subsequent queries and each query has a query interval in addition to a pattern to search for. The goal is to find only those occurrences of the pattern in the text that start in the given query interval. This problem is interesting, because, in many text search situations, one may want to search only in a part of the text, e.g. restricting the search to only parts of a long DNA sequence. To achieve an optimal query time, in [4], Problem PMQI was reduced to Problem RNV and the latter was solved with a constant query time against a data structure requiring $O(n^2)$ preprocessing time and space. It was left as an open problem to devise a better data structure without losing the constant time query capability. The goal of this paper is to present such a data structure. Notably, Problem RNV turns out to be useful in a number of other problems as well. As we will show in Section 5, Problem RNV can be used to get optimal query times for a number interesting problems studied in [7] and related to *string statistics* problem [3, 1].

It is worth-mentioning here that, despite extensive results on various range searching problems, we are not aware of any result from the literature that directly addresses Problem RNV. It seems to be possible to get a query time of $O(\log \log n)$ by using an efficient data structure for the much studied "3-sided Query" problem along with a 'persistent' data structure to 'select' the appropriate answer from the answer set of a "3-sided Query" [9]. However, our goal is to facilitate constant time query capability with a data structure requiring $o(n^2)$ time and space. In the rest of this paper, we follow the following convention adopted from [2]: if an algorithm has preprocessing time $f(n)$ and query time $g(n)$, we will say that the algorithm has complexity $\langle f(n), g(n)\rangle$.

The rest of the paper is organized as follows. In Section 2, we review the $\langle O(n^2), O(1)\rangle$ algorithm presented in [4]. In Sections 3 and 4, we present two different algorithms to solve Problem RNV with complexity $\langle O(n^{1.5}), O(1)\rangle$ and $\langle O(n^{1+\epsilon}), O(1)\rangle$ respectively. In Section 5, we discuss their possible applications.

2. The $\langle O(n^2), O(1)\rangle$ Algorithm

In this section, we briefly review the algorithm for Problem RNV (referred to as *Algorithm CIR* henceforth) presented in [4]. First, we formally define the much studied Range Minimum Query Problem, which is used by the CIR algorithm.

Problem 2.1. Range Minimum Query (Problem RMQ). We are given an array $A[1..n]$ of numbers. We need to preprocess A to answer the following form of queries:
Query: Given an interval $[\ell..r], 1 \leq \ell \leq r \leq n$, the goal is to find the minimum (maximum, in the case of Range Maximum Query) value $A[k]$ for $\ell \leq k \leq r$.

We use $RMQ_A([\ell..r])$ to denote the range minimum query on array A for the interval $[\ell..r]$. Problem RMQ has received much attention in the literature and Bender and Farach-Colton presented an algorithm with complexity $\langle O(n), O(1)\rangle$, using $O(n \log n)$-bits

of space [2][1]. Recently, Sadakane [10] presented a succinct data structure, which achieves the same time complexity using $O(n)$ bits of space. Very recently, Fischer and Heun [5] presented an algorithm with the same time complexity requiring optimal $2n + o(n)$ bits of additional space.

2.1. Algorithm CIR

Algorithm CIR maintains n arrays $B_i, 1 \leq i \leq n$. Each array B_i has n elements. So, B could be viewed as a two dimensional array. Algorithm CIR fills each array B_i depending on A as follows. For each $1 \leq i \leq n$ it stores in B_i the difference between i and the corresponding element of A, and then replace all negative entries of B_i with ∞. More formally, for each $1 \leq i \leq n$ and for each $1 \leq j \leq n$, algorithm CIR sets $B_i[j] = A[j] - i$, if $A[j] \geq i$; otherwise it sets $B_i[j] = \infty$. Then, each $B_i, 1 \leq i \leq n$, is preprocessed for the RMQ problem. This completes the construction of the data structure. It is clear that, Algorithm CIR requires $O(n^2)$ preprocessing time. The query processing is as follows. Consider the query $RNV_A([\ell..r], \mathcal{K})$. Then, we simply need to apply range minimum query in $B_{\mathcal{K}}$ for the interval $[\ell..r]$, i.e., we need to execute the query: $RMQ_{B_{\mathcal{K}}}([\ell..r])$. This gives us the following theorem.

Theorem 2.2. [4]. *For Problem RNV, we can construct a data structure in $O(n^2)$ time and space to answer the relevant queries in $O(1)$ time per query.*

3. An Improved Algorithm with Complexity $\langle O(n^{1.5}), O(1)\rangle$

In this section, we present an algorithm that improves on Algorithm CIR. In what follows, we use the following notations. Given an array $A[1..n]$, we denote by $\widehat{A}$, the underlying set comprising of all the (distinct) elements of A. In other words, $\widehat{A} = \{A[i] \mid 1 \leq i \leq n\}$. We define $\min(A) = A[i]$, such that $A[i] \leq A[j]$ for all j in $[1..n]$. Given a sub-array $A[\ell..r], 1 \leq \ell \leq r \leq n$, of the array A, we further define $left(A[\ell..r]) = \ell$ and $right(A[\ell..r]) = r$. We say that, a range $[\ell..r]$ is *nonexistent*, if $\ell > r$; otherwise, $[\ell..r]$ is said to be *existent*. Furthermore, given a range $[\ell..r], 1 \leq \ell \leq r \leq n$, and a sub-array $A[i..j], 1 \leq i \leq j \leq n$ of an array $A[1..n]$, we say that the range $[\ell..r]$ is *confined* in the sub-array $A[i..j]$, if, and only if, we have $i \leq \ell \leq r \leq j$. Now, recall that, our goal is to construct a data structure requiring $o(n^2)$ time and space without losing the constant time query capability. Below we present the idea we employ.

In this section, we will assume that, we are looking for the immediate higher value (instead of 'equal or higher') than the given value $\mathcal{K}$ in Problem RNV. It is easy to realize that, it doesn't really create any problem for the actual case.

In the first phase, we divide the array $A[1..n]$ into $\lceil n/\wp \rceil = q$ sub-arrays $D_j, 1 \leq j \leq q$. Now, we add the number 0 to the beginning of each $D_j, 1 \leq j \leq q$. It is easy to realize that, each D_j has exactly $\wp + 1$ elements except possibly the last one, which may have less. Now, we apply a slight variation of Algorithm CIR on each $D_j, 1 \leq j \leq q$ as follows. For each D_j, we maintain $|D_j|$ arrays, $B_j^{\ell}[1..|D_j|], \ell \in D_j$. Notably, the naming convention followed

[1]The same result was achieved in [6], albeit with a more complex data structure.

for the B_j^ℓ arrays are for better exposition. For example, if $D_j = \langle 0, 1, 9, 2, 6\rangle$, then we have $B_j^0, B_j^1, B_j^9, B_j^2$ and B_j^6. Now, we fill each such $B_j^\ell[1..|D_j|]$ as follows:

$$B_j^\ell[i] = \begin{cases} D_j[i] & \text{If } D_j[i] > \ell \\ \infty & \text{Otherwise} \end{cases} \tag{3.1}$$

In the second phase, we construct q arrays $E_i[0..n], 1 \le i \le q$. E_i is filled up as follows:

$$E_i[j] = \begin{cases} j & \text{If } j \in D_i \\ E_i[j-1] & \text{Otherwise}^a \end{cases} \tag{3.2}$$

[a]Recall that $0 \in D_i$ for all $1 \le i \le q$.

In the third phase, we construct n arrays $F_k[1..q], 1 \le k \le n$, where we fill:

$$F_k[i] = \min\{D_i[j] : D_i[j] > k \text{ and } 1 \le j \le |D_i|\} = RNV_{D_i}([1..|D_i|], k)$$

Please note, that all the F_k arrays can be computed in $O(nq)$ time. Finally, we preprocess each F_k $(1 \le k \le n)$ and all B_j^ℓ arrays for the RMQ problem. This completes the construction of our data structure. In what follows, we use RNV_DS1 to refer to this data structure.

Algorithm 1 Function $RNV\_Query(A[\ell..r], \mathcal{K}))$

```
1: let ℓ' = (i1 − 1) · ℘ < ℓ ≤ i1 · ℘
2: let r' = (i2 − 1) · ℘ < r ≤ i2 · ℘
3: if i1 = i2 then
4:     {ℓ and r are in the same block}
5:     Set j = i1, u = E_{i1}[K]
6:     return RMQ_{B_j^u}([(ℓ − ℓ')..(r − r')])
7: else
8:     Set val1 = val2 = val3 = ∞.
9:     Set u1 = E_{i1}[K], u2 = E_{i2}[K].
10:    Set val1 = RMQ_{B_{i1}^{u1}}([(ℓ − ℓ')..|Du|]){Executing RNV_{D_{i1}}([(ℓ − ℓ')..|D_{i1}|], K}
11:    Set val3 = RMQ_{B_{i2}^{u2}}([1..(r − r')]){Executing RNV_{D_{i2}}([1..(r − r')], K}
12:    if i2 − i1 > 1 then
13:        Set val2 = RMQ_{F_K}([(i1 + 1)..(i2 − 1)])
14:    end if
15:    return min{val1, val2, val3}
16: end if
```

3.1. Query Processing

In this section, we discuss the query processing. Suppose, we are considering the following query: $RNV_A([\ell..r], \mathcal{K})$. We compute, ℓ', r', i_1 and i_2, such that, $\ell' = (i_1-1)\cdot\wp < \ell \le i_1\cdot\wp$ and $r' = (i_2 - 1) \cdot \wp < r \le i_2 \cdot \wp$. Then, we can divide the range $[\ell..r]$ into 3 consecutive ranges, namely $[\ell..i_1 \times \wp], [i_1 \times \wp + 1..(i_2 - 1) \times \wp]$ and $[(i_2 - 1) \times \wp + 1..r]$ (See Figure 1). Now, we proceed with the query processing as follows. We have the following cases.

Case 1: $i_1 = i_2$**:** In this case, the range $[\ell..r]$ is in the same sub-array D_{i_1}. So, we only perform the following RMQ query, the answer of which is returned as the desired result: $RNV_{D_{i_1}}([(\ell - \ell')..(r - r')], \mathcal{K}) = RMQ_{B_{i_1}^{E_{i_1}[\mathcal{K}]}}([(\ell - \ell')..(r - r')])$.

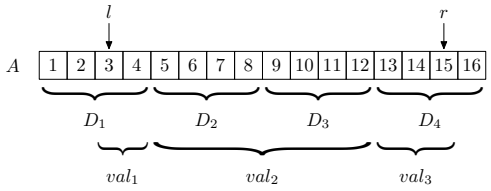


Figure 1: The situation of an RNV query

Case 2: $i_2 > i_1$**:** In this case, we first initialize val_1, val_2 and val_3 to ∞ and then we proceed with the query processing as follows. We first perform the following RMQ queries:

$$val_1 = RNV_{D_{i_1}}([(\ell - \ell')..|D_{i_1}|], \mathcal{K}) = RMQ_{B_{i_1}^{E_{i_1}[\mathcal{K}]}}([(\ell - \ell')..|D_{i_1}|]) \tag{3.3}$$

$$val_3 = RNV_{D_{i_2}}([1..(r - r')], \mathcal{K}) = RMQ_{B_{i_1}^{E_{i_2}[\mathcal{K}]}}([1..(r - r')]) \tag{3.4}$$

Then, if we have $i_2 - i_1 > 1$, then we perform the following RMQ query:

$$val_2 = RMQ_{F_{\mathcal{K}}}([(i_1 + 1)..(i_2 - 1)]) \tag{3.5}$$

Finally, we return the minimum of val_1, val_3 and val_2 as the final result.

3.2. Correctness and Running Time

In this section, we discuss the correctness of the above algorithm and its running time.

Lemma 3.1. *With the data structure RNV_DS1, we can correctly answer any queries of the form* $RNV_{D_i}([\ell..r], \mathcal{K})$, $1 \leq \ell \leq r \leq |D_i|$, $1 \leq \mathcal{K} \leq n$, $1 \leq i \leq q$.

Proof. Recall that, $RNV_{D_i}([\ell..r], \mathcal{K})$ is executed by calculating $RNV_{D_i}([\ell..r], E_i[\mathcal{K}])$, which in turn, is executed by performing the query $RMQ_{B_i^{E_i[\mathcal{K}]}}([\ell..r])$. From the correctness of Algorithm CIR, it is clear that, $RNV_{D_i}([\ell..r], E_i[\mathcal{K}]) \equiv RMQ_{B_i^{E_i[\mathcal{K}]}}([\ell..r])$. So it remains to show that $RNV_{D_i}([\ell..r], \mathcal{K}) \equiv RNV_{D_i}([\ell..r], E_i[\mathcal{K}])$. This is shown as follows. Recall that, by definition, if $\mathcal{K} \in D_i$ then $E_i[\mathcal{K}] = \mathcal{K}$. So, if $\mathcal{K} \in D_i$, we are done. Therefore, assume otherwise. Now, in this case, $E_i[\mathcal{K}]$ is the nearest smaller value of $\mathcal{K}$ in D_i. For the values $v < \min(\widehat{D_i} \setminus 0)$, this is ensured by the addition of 0 in each D_i. Therefore, it is easy to realize that, $RNV_{D_i}([\ell..r], E_i[\mathcal{K}])$ would return the same value as returned by $RNV_{D_i}([\ell..r], \mathcal{K})$ and hence, the lemma follows. ∎

Theorem 3.2 (Correctness)**.** *With the data structure RNV_DS1, we can correctly answer any query of the form* $RNV_A([\ell..r], \mathcal{K}), 1 \leq \ell \leq r \leq n, 1 \leq \mathcal{K} \leq n$.

Proof. Recall that, the range $[\ell..r]$ is transformed into (up to) 3 consecutive ranges, namely $r_1 \equiv [\ell..i_1 \times \wp], r_2 \equiv [i_1 \times \wp + 1..(i_2 - 1) \times \wp]$ and $r_3 \equiv [(i_2 - 1) \times \wp + 1..r]$. Now, the range r_1 (resp. r_3) is confined within the sub-array D_{i_1} (resp. D_{i_2}). On the other hand, if r_2 is existent, then it can span over one or more sub-arrays, $D_{i_1+1}, \ldots, D_{i_2-1}$ and, in that case, it completely contains those sub-arrays, i.e. $i_1 \times \wp + 1 = left(D_{i_1+1})$ and $(i_2 - 1) \times \wp = right(D_{i_2-1})$. It is clear that, the minimum of the results of the corresponding RNV queries in the three ranges, namely, r_1, r_2 and r_3, is the final result. Now, recall that, we have the following two cases.

case 1: $i_1 = i_2$**:** It is easy to verify that, this case arises when the range $[\ell..r]$ is confined in the sub-array D_{i_1}. Therefore, it is easy to verify that, we have $RNV_A([\ell..r], \mathcal{K}) \equiv RNV_{D_{i_1}}([(\ell - \ell')..(r - r')], \mathcal{K})$, and by Lemma 3.1, we get the correct result.

case 2: $i_2 > i_1$**:** It is easy to see that, if we have $i_2 - i_1 > 1$, then all 3 intervals are existent; otherwise, r_2 is non-existent. Now, recall that, we initialize val_1, val_3 and val_2 to ∞. Since, both the ranges r_1 and r_3 are confined in the sub-arrays D_{i_1} and D_{i_2} respectively, by Lemma 3.1, the two corresponding queries, namely, Queries 3.3 and 3.4 are correctly executed and the results are stored in val_1 and val_3.

Now, assume that the range r_3 is existent and that, $v_i = RNV_{D_i}([1..|D_i|], \mathcal{K}), i \in [i_1 + 1..i_2 - 1]$. Then, it is easy to verify that:

$$RNV_A([\ell..r], \mathcal{K}) = \min(val_1, val_3, \min_{i \in [i_1+1..i_2-1]}(v_i)).$$

Now, we return $\min(val_1, val_3, val_2)$ as the answer. Hence, it suffices to show that $val_2 = \min_{i \in [i_1+1..i_2-1]}(v_i)$. Recall that, val_2 is evaluated according to Equation 3.5. By definition, each entry of $F_{\mathcal{K}}$ correctly (Lemma 3.1) stores the result of the RNV query for the value $\mathcal{K}$ and for the whole range for the corresponding sub-array. Therefore, the range minimum query does provide us with the desired value.

Finally, if r_2 is nonexistent, then val_2 remains assigned to ∞. Therefore, the result returned, i.e., the minimum of val_1, val_3 and val_2, is correct.

■

Theorem 3.3. *The data structure RNV_DS1 can be constructed in $O(n\wp + n^2/\wp)$ time.*

Proof. We deduce the construction time of RNV_DS1 phase by phase as follows.

Phase 1: Each sub-array $D_j, 1 \leq j \leq q = \lceil n/\wp \rceil$ has at most $\wp+1$ elements. It is easy to see that, the application of the (slight variation of) Algorithm CIR requires $O(\wp^2)$ time per sub array. Therefore, in total, time required by Phase 1 is $O(\wp^2) \times q = O(\wp^2) \times \lceil n/\wp \rceil = O(n\wp)$ in the worst case.

Phase 2: Initializing and filling up the arrays $E_i[0..n], 1 \leq i \leq q$ requires $O(n) \times q = O(n^2/\wp)$ time.

Phase 3: In this phase, we construct the arrays $F_k[1..q]$, for $1 \leq k \leq n$. This can easily be done in $O(nq) = O(n^2/\wp)$ time. We also preprocess arrays F_k and B_j^l for the RMQ queries, what requires also $O(nq) = O(n^2/\wp)$ time.

Therefore, in total, the time required for the construction of RNV_DS1 is $O(n\wp)+O(n^2/\wp)+O(n^2/\wp) = O(n\wp + n^2/\wp)$. ■

Corollary 3.4. *The data structure RNV_DS1 can be constructed in $O(n^{1.5})$ time.*

Proof. This can be achieved if we assume that $\wp = \sqrt{n}$. ■

Theorem 3.5. *Given the data structure RNV_DS1, we can answer the RNV queries in $O(1)$ time per query.*

Proof. It is clear that, given RNV_DS1, an RNV query is answered by executing up to 2 RNV queries on the sub-arrays and possibly 1 RMQ queries on the appropriate F array. Each of these queries requires $O(1)$ time. Therefore, the theorem follows. ■

4. An Improved Algorithm with Complexity $\langle O(n^{1+\epsilon}), O(1)\rangle$

In this section, we present a different algorithm for problem RNV by taking a slightly different approach. We start with a slightly different $\langle O(n^2), O(1)\rangle$ algorithm and present a new algorithm built on top it. This algorithm follows a similar strategy as algorithm CIR and is referred to as the *base algorithm* henceforth.

4.1. The Base Algorithm:

We define arrays $B_j, 1 \le j \le n$ as follows:

$$B_j[i] = \begin{cases} A[i] & \text{if } A[i] \ge j \\ \infty & \text{if } A[i] < j \end{cases}$$

Now, the preprocessing is done as follows.

1: **for** $j = 1, \ldots, n$ **do**
2: Preprocess sequence B_j for Problem RMQ
3: **end for**

After the above data structure is constructed, we can perform the queries as follows. Similar to what was done in algorithm CIR, given the query $RNV_A([\ell..r], \mathcal{K})$, we just return $RMQ_{B_\mathcal{K}}([l..r])$. It is easy to see that the base algorithm is correct and its running time is $\langle O(n^2), O(1)\rangle$. In the rest of this section, we present an improved algorithm based on the base algorithm.

4.2. Improved algorithm

In this section we describe a method for improving preprocessing time of any RNV algorithm, the cost paid for the improvement is slight (namely $O(1)$) increase of the RNV query time. Suppose, we are given the array A of length n, the parameter $\wp$, and an algorithm RNVALG for Problem RNV with complexity $\langle f(n), g(n)\rangle$. We will show how to improve the preprocessing time of RNVALG.

In the first phase we divide possible values of parameter $\mathcal{K}$, into $\lceil n/\wp \rceil = q$ interval sets K_j, where $K_j = \{i : (j-1)\cdot\wp < i \le j\cdot\wp\}$. For each j $(1 \le j \le q)$ we compute following arrays:

- array B'_j ($|B'_j| = n$) — containing information about elements of array A strictly larger than $(j-1)\cdot\wp$

$$B'_j[i] = \begin{cases} A[i] & \text{if } A[i] > (j-1)\cdot\wp \\ \infty & \text{otherwise} \end{cases}$$

- set $\mathcal{C}_j = \{i : A[i] \in K_j\}$ — containing indices of the elements of array A with values from the range K_j; by C_j we will denote the array consisting of elements of $\mathcal{C}_j$ sorted in the ascending order, $|C_j| \le \wp$,
- array D_j ($|D_j| \le \wp$) — contains the elements of array A from the range K_j, in the order as they appear in A; each element is decreased by $(j-1)\cdot\wp$, to ensure that the array D_j is a permutation of $\{1..|D_j|\}$:

$$D_j[i] = A[C_j[i]] - (j-1)\cdot\wp, \quad \text{for } 1 \le i \le |C_j|$$

- array E_j ($|E_j| = n+1$) — containing indices used for translating queries from array A_j to array D_j; $E_j[i]$ denotes the number of elements from $A[1..i]$ from the range K_j:

$$E_j[i] = \begin{cases} E_j[i-1]+1 & \text{if } A[i] \in K_j \text{ and } i > 0 \\ E_j[i-1] & \text{if } A[i] \notin K_j \text{ and } i > 0 \\ 0 & \text{if } i = 0 \end{cases}$$

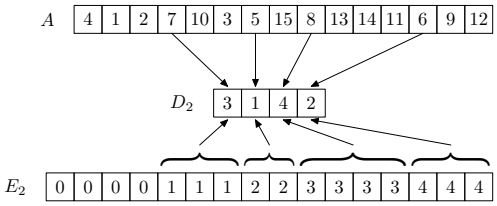


Figure 2: Example of computing arrays D_j and E_j, for $n = 15$, $\wp = 4$, $j = 2$, $\mathcal{K}_j = [5..8]$

Then the algorithm preprocesses each array B'_j for range minimum queries, and each array D_j for range next value queries (using RNVALG).

Algorithm 2 Construction of RNV_DS2($\wp$,RNVALG)

```
1: for j = 1, ..., ⌈n/℘⌉ do
2:     compute arrays B'_j, C_j, D_j, E_j,
3:     preprocess sequence B'_j for RMQ (Range Minimum Queries)
4:     preprocess sequence D_j for RNV (Range Next Value Queries) using RNVALG
5: end for
```

The B'_j arrays will be used for answering the range next value queries if the answer is outside of the range K_j. The D_j will be used if the answer is within the range K_j. Since we do not know in the advance which case is valid, the algorithm tries both cases, and then chooses the smaller result.

Algorithm 3 Query Processing of RNV_DS2($\wp$,RNVALG)

```
1: Set a_1 = a_2 = ∞
2: Set j, such that: x = (j − 1) · ℘ < K ≤ j · ℘
3: if j < q then
4:     a_1 = RMQ_{B'_{j+1}}([ℓ..r])
5: end if
6: Set ℓ' = E_j[ℓ − 1] + 1; r' = E_j[r]
7: if ℓ' ≤ r' then
8:     a_2 = RNV_{D_j}([ℓ'..r']], K − x) + x {using algorithm RNVALG}
9: end if
10: return min(a_1, a_2)
```

Theorem 4.1. *If we are given the $\langle f(n), g(n)\rangle$ RNV algorithm, then using the RNV_DS2, we can construct $\langle O((n^2 + nf(\wp))/\wp),\ g(\wp) + O(1)\rangle$ algorithm for RNV.*

Proof. The preprocessing of the RNV_DS2 requires:

- computing $n/\wp$ arrays B'_j (each of length n), this step requires $O(n^2/\wp)$ time,

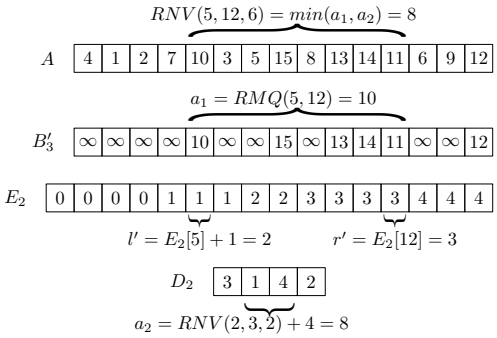


Figure 3: RNV_DS2 processing query $RNV(5, 12, 6)$ (assuming $\wp = 4$)

- preprocessing $n/\wp$ arrays B'_j for the Range Minimum Queries, this step also requires $O(n^2/\wp)$ time,
- computing $n/\wp$ arrays D_j (each of length $\wp$), clearly this step requires $O(n)$ time,
- preprocessing $n/\wp$ arrays D_j for the Range Next Value Queries using $\langle f(n), g(n)\rangle$ algorithm, this step requires $O(f(\wp) \cdot n/\wp)$ time,
- computing $n/\wp$ arrays E_j (each of length $n + 1$), this step requires $O(n^2/\wp)$ time.

The total preprocessing time is $O((n^2 + nf(\wp))/\wp)$.

Answering the Range Next Value queries requires:

- one range minimum query on the B'_j array, what can be done in $O(1)$ time,
- one recursive call of the range next value query for D_j array using $\langle f(n), g(n)\rangle$ RNV algorithm, requiring $g(\wp)$ time,
- constant number of additional operations (i.e. accessing arrays E_j)

Clearly the total query time is $g(\wp) + O(1)$. ∎

Corollary 4.2. *RNV_DS2 can be constructed in $O(n^{1.5})$ running time and space.*

Proof. This can be achieved if we use the RNV_DS2 construction method, with $\wp = \sqrt{n}$, and using as RNVALG, the base algorithm (with complexity $\langle O(n^2), O(1)\rangle$). ∎

We can obtain even more efficient algorithm, carefully iterating RNV_DS2 construction.

Theorem 4.3. *For any given positive constant $\epsilon > 0$, we can construct $\langle O(n^{1+\epsilon},\ O(1)\rangle$ algorithm form RNV using the RNV_DS2.*

Proof. Let RNV_DS2(0) denote the base algorithm for RNV (with the complexity $\langle O(n^2), O(1)\rangle$). For any $i > 0$, let RNV_DS2(i) denote the algorithm obtained using RNV_DS2 with RNVALG = RNV_DS2($i-1$) and $\wp = n^{\frac{i}{i+1}}$. From the theorem 4.1 the RNV_DS2(1) has the complexity $\langle O(n^{1.5}), O(1)\rangle$, the RNV_DS2(2) has the complexity $\langle O(n^{1+\frac{1}{3}}), O(1)\rangle$. By simple induction, one can easily prove, that the RNV_DS2(i) has the complexity $\langle O(n^{1+\frac{1}{i}}), O(i)\rangle$. ∎

5. Applications

In this section, we discuss possible applications of Problem RNV. As has already been mentioned in Section 1, the study of the RNV problem in [4] was motivated by Problem

PMQI, a variant of the classic pattern matching problem. Problem PMQI is formally defined as follows (We use $Occ_{\mathcal{T}}^{\mathcal{P}}$ to denote the occurrence set for the classic pattern matching problem):

Problem 5.1. Pattern Matching in a Query Interval (Problem PMQI). Suppose we are given a text $\mathcal{T}$ of length n. Preprocess $\mathcal{T}$ to answer queries of the following form.
Query: We are given a pattern $\mathcal{P}$ of length m and a query interval $[\ell..r]$, with $1 \leq \ell \leq r \leq n$. Let us denote by $Occ_{\mathcal{T}}^{\mathcal{P}}$ the set of all occurrences of $\mathcal{P}$ in $\mathcal{T}$. We are to construct the set:

$$Occ_{\mathcal{T}[\ell..r]}^{\mathcal{P}} = \{i \mid i \in Occ_{\mathcal{T}}^{\mathcal{P}} \text{ and } i \in [\ell..r]\}$$

Using the reduction of [4] from Problem PMQI to Problem RNV, we obtain the following theorem.

Theorem 5.2. *We can construct a data structure for Problem PMQI in* $O(\max(n^{1+\epsilon}, n \log \sigma))$ *time and* $O(n^{1+\epsilon})$ *space, and we can answer the relevant queries in the optimal* $O(m + |Occ_{\mathcal{T}[\ell..r]}^{\mathcal{P}}|)$ *time per query.*

A more general problem called PMI was also handled in [4].

Problem 5.3. Generalized Pattern Matching with Intervals (Problem PMI). Suppose we are given a text $\mathcal{T}$ of length n and a set of intervals $\pi = \{[s_1..f_1], [s_2..f_2], \ldots, [s_{|\pi|}..f_{|\pi|}]\}$, such that $s_i, f_i \in [1..n]$ and $s_i \leq f_i$, for all $1 \leq i \leq |\pi|$. Preprocess $\mathcal{T}$ to answer queries of the following form.
Query: Given a pattern $\mathcal{P}$ and a query interval $[\ell..r]$, such that $\ell, r \in [1..n]$ and $\ell \leq r$, construct the set

$$Occ_{\mathcal{T}[\ell..r],\pi}^{\mathcal{P}} = \{i \mid i \in Occ_{\mathcal{T}}^{\mathcal{P}} \text{ and } i \in [\ell, r] \cap \varpi \text{ for some } \varpi \in \pi\}$$

To solve Problem PMI, a data structure with $O(n \log^3 n)$ time, $O(n \log^2 n)$ space was constructed in [4]; the query time achieved was $O(m + \log \log n + |Occ_{\mathcal{T}[\ell..r],\pi}^{\mathcal{P}}|)$. It was left as an open problem to achieve the optimal query time for Problem PMI [4]. Interestingly, using Problem RNV, we can get the optimal query time for Problem PMI as well. The details are left for the journal version; but we report the new result in the following theorem.

Theorem 5.4. *For problem PMI, we can construct a data structure in* $O(\max(n^{1+\epsilon}, n \log \sigma))$ *time and* $O(n^{1+\epsilon})$ *space, and we can answer the relevant queries in the optimal* $O(m + |Occ_{\mathcal{T}[\ell..r],\pi}^{\mathcal{P}}|)$ *time per query.*

In the rest of this section, we consider three recent variants of the classic pattern matching problem, which we define below after defining some related concepts. Given two occurrences $i, j \in [1..n - m + 1], j > i$ of a pattern $\mathcal{P}[1..m]$ in a text $\mathcal{T}[1..n]$, we say that j is *minimal* with respect to i, if, and only if, there exists no occurrence of $\mathcal{P}$ in $\mathcal{T}$ in the range $[i + 1..j - 1]$. And, two occurrences $i, j \in [1..n - m + 1]$ of $\mathcal{P}$ in $\mathcal{T}$ are said to be *non-overlapping*, if, and only if, $|j - i| \geq m$. Otherwise, they are said to be overlapping.

Problem 5.5. Suppose we are given a text $\mathcal{T}$ of length n. Preprocess $\mathcal{T}$ to answer the following form of queries:
Query: Given a pattern $\mathcal{P}$ of length m, and an index i, we want to find out an occurrence $i' \geq i$ of $\mathcal{P}$ in $\mathcal{T}$, such that i' is minimal with respect to i.

Problem 5.6. Suppose we are given a text $\mathcal{T}$ of length n. Preprocess $\mathcal{T}$ to answer the following form of queries:

Query: Given a pattern $\mathcal{P}$ of length m, and a list of indices $\mathcal{U} = \langle i_1, \ldots, i_\ell \rangle$, our goal is to construct the list $\mathcal{V} = \langle j_1, \ldots, j_\ell \rangle$, such that, for all $k \in [1..\ell]$, j_k is an occurrence of $\mathcal{P}$ in $\mathcal{T}$ and $j_k \in \mathcal{V}$ is, either minimal with respect to $i_k \in \mathcal{U}$ or equal to $Nill$. The latter case means that there doesn't exist any occurrence to the right of i_k.

Problem 5.7. Suppose we are given a text $\mathcal{T}$ of length n. Preprocess $\mathcal{T}$ to answer the following form of queries:
Query: Given a pattern $\mathcal{P}$ of length m, and an interval $[i..j]$, we want to find an ascending sequence $\mathcal{U} = \langle i_1, \ldots, i_\ell \rangle$ of non-overlapping occurrences of $\mathcal{P}$ in $\mathcal{T}$, such that $i \leq i_1 \leq i_\ell \leq j$ and ℓ is maximal.

Problems 5.5 to 5.7 were handled very recently in [7]. The corresponding data structures presented in [7] for the above problems requires $O(n \log n)$ storage and $O(n \log n \log \log n)$ expected preprocessing time each. The query time achieved in [7], for Problem 5.6 and 5.7 is $O(m + \ell \log \log n)$ and for Problem 5.5 is $O(m + \log \log n)$. Notably, none of the query times achieved in [7] are optimal. In the rest of this section, we briefly show, how Problem RNV can be used to achieve optimal query times for the above problems. We remark however that, we omit many of the details for space constraint and left them for the journal version.

5.1. Problems 5.5 and 5.6

It is clear that, Problem 5.5 is a simpler version of the Problem 5.6. Interestingly, we can use Problem RNV to solve both the problems efficiently. We first consider Problem 5.5. Following the techniques of [4], we construct a suffix tree and do some preprocessing on it to get $Occ_T^{\mathcal{P}}$ implicitly in the form of an array $\mathcal{L}$ and an interval $[a..b]$. More specifically, using the techniques of [4], after the preprocessing, we can implicitly have $Occ_T^{\mathcal{P}}$ in $\mathcal{L}[a..b]$ in $O(m)$ time. Now, it is easy to see that to solve the query of problem 5.5, we simply need to get the answer of the following query:

$$RNV_{\mathcal{L}}([a..b], i) \tag{5.1}$$

Therefore, we have the following result.

Theorem 5.8. *For Problem 5.5, we can construct a data structure in $O(\max(n^{1+\epsilon}, n \log \sigma))$ time and $O(n^{1+\epsilon})$ space, and we can answer the relevant queries in the optimal $O(m)$ time per query.*

Proof. For the preprocessing, we first construct the suffix tree and do the preprocessing of [4], requiring $O(n \log \sigma)$ time, where $\sigma = \min(n, |\Sigma|)$. Then we preprocess $\mathcal{L}$ for Problem RNV. Total construction time and space complexity is, $O(\max(n^{1+\epsilon}, n \log \sigma))$ and $O(n^{1+\epsilon})$ respectively. As for the query, we require $O(m)$ time to get $Occ_T^{\mathcal{P}}$ implicitly [4]. Then, we just need to perform the Query 5.1 requiring constant time. Hence, the result follows. ∎

We can easily extend the above result for Problem 5.6, simply by executing RNV queries, $RNV_{\mathcal{L}}([a..b], i)$ for all $i \in \mathcal{U}$. Therefore, we get the following theorem.

Theorem 5.9. *For Problem 5.6, we can construct a data structure in $O(\max(n^{1+\epsilon}, n \log \sigma))$ time and $O(n^{1+\epsilon})$ space, and we can answer the relevant queries in the optimal $O(m + \ell)$ time per query.*

5.2. Problem 5.7

To solve Problem 5.7, we follow the greedy strategy of [7] as follows. Suppose, we have the set $Occ_{\mathcal{T}}^{\mathcal{P}}$ in the list $\mathcal{W} = \langle i_1, \ldots, i_{|Occ_{\mathcal{T}}^{\mathcal{P}}|} \rangle$ in ascending order. Now, we construct another list $\mathcal{Y}$ as follows. We first put i_1 in $\mathcal{Y}$. We use $last(\mathcal{Y})$ to denote the most recently put index in $\mathcal{Y}$. Now we scan the list $\mathcal{W}$ from left to right and put $i_k \in \mathcal{W}$ in $\mathcal{Y}$, only if i_k and $last(\mathcal{Y})$ are non-overlapping. It was proved in [7] that, $|\mathcal{Y}|$ is maximal. Therefore, we have the following theorem.

Theorem 5.10. *For Problem 5.7, we can construct a data structure in* $O(\max(n^{1+\epsilon}, n \log \sigma))$ *time and* $O(n^{1+\epsilon})$ *space, and we can answer the relevant queries in the optimal* $O(m + \ell)$ *time per query.*

Proof. We do the same preprocessing as we did for Problems 5.5 and 5.6 and hence achieve the same preprocessing time and space complexity. Now, we consider the query. We start with the query $RNV_{\mathcal{L}}([a..b], i + 1)$. Now suppose, the query returns q. Now, if $q \leq j$, then we put q in $\mathcal{U}$ and perform the query $RNV_{\mathcal{L}}([a..b], q + m)$ and continue as before. We stop when we get a query result q' such that $q' > j$. It is easy to verify that this would correctly construct a maximal list $\mathcal{U}$. Finally, since each of the queries require constant time, the result follows. ∎

References

[1] A. Apostolico and F. P. Preparata. Data structures and algorithms for the string statistics problem. *Algorithmica*, 15(5):481–494, 1996.

[2] M. A. Bender and M. Farach-Colton. The lca problem revisited. In G. H. Gonnet, D. Panario, and A. Viola, editors, *Latin American Theoretical INformatics (LATIN)*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000.

[3] G. S. Brodal, R. B. Lyngsø, A. Östlin, and C. N. S. Pedersen. Solving the string statistics problem in time $O(n \log n)$. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 728–739. Springer, 2002.

[4] M. Crochemore, C. S. Iliopoulos, and M. S. Rahman. Finding patterns in given intervals. In A. Kucera and L. Kucera, editors, *MFCS*, volume 4708 of *Lecture Notes in Computer Science*, pages 645–656. Springer, 2007.

[5] J. Fischer and V. Heun. A new succinct representation of rmq-information and improvements in the enhanced suffix array. In B. Chen, M. Paterson, and G. Zhang, editors, *ESCAPE*, volume 4614 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2007.

[6] H. Gabow, J. Bentley, and R. Tarjan. Scaling and related techniques for geometry problems. In *Symposium on the Theory of Computing (STOC)*, pages 135–143, 1984.

[7] O. Keller, T. Kopelowitz, and M. Lewenstein. Range non-overlapping indexing and successive list indexing. In F. K. H. A. Dehne, J.-R. Sack, and N. Zeh, editors, *WADS*, volume 4619 of *Lecture Notes in Computer Science*, pages 625–636. Springer, 2007.

[8] V. Mäkinen and G. Navarro. Position-restricted substring searching. In J. R. Correa, A. Hevia, and M. A. Kiwi, editors, *LATIN*, volume 3887 of *Lecture Notes in Computer Science*, pages 703–714. Springer, 2006.

[9] E. Porat. Private communication.

[10] K. Sadakane. Succinct data structures for flexible text retrieval systems. *Journal of Discrete Algorithms*, 5(1):12–22, 2007.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 217-228
www.stacs-conf.org

CONNECTING POLYGONIZATIONS VIA STRETCHES AND TWANGS

MIRELA DAMIAN [1], ROBIN FLATLAND [2], JOSEPH O'ROURKE [3],
AND SUNEETA RAMASWAMI [4]

[1] Dept. of Computer Science, Villanova Univ., Villanova, PA 19085, USA.
E-mail address: mirela.damian@villanova.edu

[2] Dept. of Computer Science, Siena College, Loudonville, NY 12211, USA.
E-mail address: flatland@siena.edu

[3] Dept. of Computer Science, Smith College, Northampton, MA 01063, USA.
E-mail address: orourke@cs.smith.edu

[4] Dept. of Computer Science, Rutgers University, Camden, NJ 08102, USA.
E-mail address: rsuneeta@camden.rutgers.edu

Abstract. We show that the space of polygonizations of a fixed planar point set S of n points is connected by $O(n^2)$ "moves" between simple polygons. Each move is composed of a sequence of atomic moves called "stretches" and "twangs". These atomic moves walk between weakly simple "polygonal wraps" of S. These moves show promise to serve as a basis for generating random polygons.

1. Introduction

This paper studies polygonizations of a fixed planar point set S of n points. Let the n points be labeled p_i, $i = 0, 1, \ldots, n-1$. A *polygonization* of S is a permutation σ of $\{0, 1, \ldots, n-1\}$ that determines a polygon: $P = P_\sigma = (p_{\sigma(0)}, \ldots, p_{\sigma(n-1)})$ is a simple (non-self-intersecting) polygon. We will abbreviate "simple polygon" to *polygon* throughout. We do not make any general position assumptions about S, except to assume the points do not lie in one line so that there is at least one polygon whose vertex set is S. A point set S may have as few as 1 polygonization, if S is in convex position,[1] and as many as $2^{\Theta(n)}$ polygonizations. For the latter, see Fig. 1a and [CHUZ01] for additional details.

Our goal in this work is to develop a computationally natural and efficient method to explore all polygonizations of a fixed set S. One motivation is the generation of "random polygons" by first generating a random S and then selecting uniformly at random a polygonization of S. Generating random polygons efficiently is a long unsolved problem; only heuristics [AH96] or algorithms for special cases [ZSSM96], [HHH02] are known. Our work can be viewed as following a suggestion in [ZSSM96]:

1998 ACM Subject Classification: Nonnumerical Algorithms: F.2.2; Discrete Mathematics: G.2.
Key words and phrases: polygons, polygonization, random polygons, connected configuration space.

[1] S is in convex position if every point in S is on the hull of S.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

> "start with a ... simple polygon and apply some simplicity-preserving, reversible operations ... with the property that any simple polygon is reachable by a sequence of operations"

Our two operations are called *stretch* and *twang* (defined in Sec. 2.2). Neither is simplicity preserving, but they are nearly so in that they produce polygonal wraps defined as follows.

Definition 1.1. *A* polygonal wrap $\mathcal{P}_\sigma$ *is determined by a sequence* σ *of point indices that includes every index in* $\{0, 1, \ldots, n-1\}$ *at least once, such that there is a perturbation of the points in multiple contact that renders* $\mathcal{P}_\sigma$ *a simple closed curve through the perturbed points in* σ *order.*

Thus polygonal wraps disallow proper crossings[2] but permit self-touching. This notion is called a "weakly simple polygon" in the literature, but we choose to use our terminology to emphasize the underlying fixed point set and the nature of our twang operation. Fig. 1b shows a polygonal wrap with five double-contacts (p_1, p_4, p_5, p_8 and p_9).

Stretches and twangs take one polygonal wrap to another. A stretch followed by a natural sequence of twangs, which we call a *cascade*, constitutes a *forward move*. Forward moves (described in Sec. 2.3) take a polygon to a polygon, i.e., they are simplicity preserving. Reverse moves will be introduced in Sec. 6. A *move* is either a forward or a reverse move. We call a stretch or twang an *atomic move* to distinguish it from the more complex forward and reverse moves.

Our main result is that the configuration space of polygonizations for a fixed S is connected by forward/reverse moves, each of which is composed of a number of stretches and twangs, and that the diameter of the space is $O(n^2)$ moves. We can bound the worst-case number of atomic moves constituting a particular forward/reverse move by the geometry of the point set. Experimental results on random point sets show that, in the practical situation that is one of our motivations, the bound is small, perhaps even constant. We have also established loose bounds on the worst-case number of atomic operations as a function of n: an exponential upper bound and a quadratic lower bound. Tightening these bounds has so far proven elusive and is an open problem.

One can view our work as in the tradition of connecting discrete structures (e.g., triangulations, matchings) via local moves (e.g., edge flips, edge swaps). Our result is comparable to that in [vLS82], which shows connectivity of polygonizations in $O(n^3)$ edge-edge swap moves through intermediate self-crossing polygons, and to that in [HHH02], which established noncrossing connectivity within special classes of polygonizations. The main novelty of our work is that we avoid proper crossings but achieve connectivity via polygonal wraps. We explore the possible application to random polygons briefly in Sec. 8. For the majority of this paper, we concentrate on defining the moves and establishing connectivity.

We begin by defining pockets, which play a central role in our algorithms for polygonal transformations. Then in Sec. 2.1 we describe two natural operations that transform one polygon into another but fail to achieve connectivity of the configuration space of polygonizations, which motivates our definitions of stretches and twangs in Sec. 2.2. Following these preliminaries, we establish connectivity and compute the diameter in Secs. 3–7. We conclude with open problems in Sec. 9. Omitted proofs are in [DFOR07].

[2]Two segments properly cross if they share a point x in the relative interior of both, and cross transversely at x.

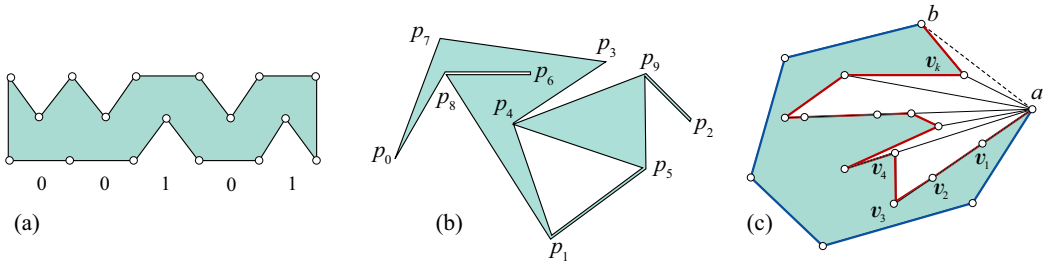


Figure 1: Examples. (a) A set of $n = 3k + 2$ points that admits 2^k polygonizations. (b) Polygonal wrap $\mathcal{P}_\sigma$ with $\sigma = (0, 8, 6, 8, 1, 5, 9, 2, 9, 4, 5, 1, 4, 3, 7)$ (c) A polygonization with one pocket with lid ab.

1.1. Pockets and Canonical Polygonization

Let P be a polygonization of S. A hull edge ab that is not on ∂P is called a *pocket lid*. The polygon external to P bounded by P and ab is a *pocket* of P. For a fixed hull edge ab, we define the *canonical polygonization* of S to be a polygon with a single pocket with lid ab in which the pocket vertices are ordered by angle about vertex a, and from closest to farthest from a if along the same line through a. We call this ordering the *canonical order* of the pocket vertices; see Fig. 1c. The existence of this canonical polygonization for any point set S not in convex position was established in [CHUZ01].

2. Polygonal Transformations

Let P be a polygon defined by a circular index sequence σ. We examine operations that permute this sequence, transforming P into a new polygon with the same set of vertices linked in a different order. Throughout the paper we use $\triangle abc$ to denote the closed triangle with corners a, b and c.

2.1. Local Transformations

The systematic study of constant-sized transformations that alter one simple polygon to another was initiated in [HHH02]. They defined a *k-flip* as an alteration of k (not necessarily consecutive) edges, and established a number of results, including showing that 3-flips are sufficient to connect polygonizations among several subclasses of polygons based on various visibility properties. But no constant k-flip move is known to be sufficient for connecting all simple polygonizations, and they conclude that "the connectivity of general simple polygons remains a challenging open problem." Although we do not resolve this open problem by a "local transformation" in their sense, we do resolve it by stepping outside their paradigm in two regards: (1) We permit polygonal wraps as intermediate structures; and (2) Our atomic moves are local and constant-sized, but they cascade into sequences of as many as $\Omega(n^2)$ atomic moves.

The most natural local transformation is a swap transposition of two consecutive vertices of P that results in a new (non-self-intersecting) polygon. A swap is a particular 2-flip. Because this is easily seen as insufficient for polygonization connectivity, 3-flips were explored in [HHH02]. Much less obviously, even these were shown to be insufficient for connectivity, except within various polygon subclasses. We review one of their 3-flips, the "planar VE-flip," which we call a Hop, because our Stretch operation is a generalization of this.

The hop operation generalizes the swap by allowing a vertex to hop to any position in the permutation, as long as the resulting polygon is simple. Fig. 2 shows the stretching of the edge ab down to vertex v, effectively "hopping" v between a and b in the permutation. We denote this operation by $\textsc{Hop}(e, v)$, where $e = ab$ (note the first argument is *from* and the second *to*).

To specify the conditions under which a hop operation is valid, we introduce some definitions, which will be used subsequently as well. A polygon P has two sides, the interior of P and the exterior of P. Let $abc = (a, b, c)$ be three noncollinear vertices consecutive in the polygonization P. We call vertex b a *true corner vertex* since the boundary of P takes a turn at b. We distinguish between the *convex side* of b, that side of P with angle $\angle abc$ smaller than π, and the *reflex side* of b, the side of P with angle $\angle abc$ larger than π. Note that this definition ignores which side is the interior and which side is the exterior of P, and so is unrelated to whether b is a convex or a reflex vertex in P. Every true corner vertex has a convex and a reflex side (collinear vertices will be discussed in Sec. 2.2). To ensure that the resulting polygon is simple, $\textsc{Hop}(e, v)$ is valid iff the following two conditions hold: (1) the triangle induced by the two edges incident to v is empty of other polygon vertices and (2) the triangle induced by e and v lies on the reflex side of v and is empty of other polygon vertices.

Although more powerful than a swap, there also exist polygons that do not admit any hops, as was established in [HHH02], and so hops do not suffice to connect all polygonizations.

The limited transformation capabilities of these 2- and 3-flip operations motivate our introduction of two new operations, *stretch* and *twang*. The former operation relaxes the two hop conditions and allows the creation of a polygonal wrap. The latter operation restores the polygonal wrap to a polygon. We show that together they are capable of transforming any polygon into a canonical form (Secs. 3-5), and from there to any other polygon (Secs. 6-7).

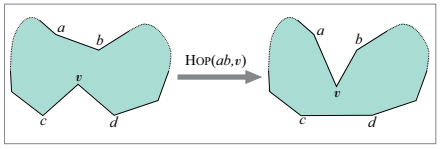


Figure 2: $\textsc{Hop}(ab, v)$ illustrated.

2.2. Stretches and Twangs

Unlike the $\textsc{Hop}(e, v)$ operation, which requires v to fully see the edge e into which it is hopping, the $\textsc{Stretch}(e, v)$ operation only requires that v see a point x in the interior[3] of e. The stretch is accomplished in two stages: (i) temporarily introduce two new "pseudovertices" on e in a small neighborhood of x (this is what we call $\textsc{Stretch}_0$ below), and (ii) remove the pseudovertices immediately using twangs.

$\textsc{Stretch}_0$. Let v see a point x in the interior of an edge e of P. By *see* we mean "clear visibility", i.e., the segment vx shares no points with ∂P other than v and x (see Fig. 3a). Note that every vertex v of P sees such an x (in fact, infinitely many x) on some e. Let x^- and x^+ be two points to either side of x on e, both in the interior of e, such that v can clearly see both x^- and x^+. Two such points always exist in a neighborhood of x. We call these points *pseudovertices*. Let $e = ab$, with x^- closer to the endpoint a of e. Then

[3]By "interior" we mean "relative interior," i.e., not an endpoint.

STRETCH$_0(e, v)$ alters the polygon to replace e with (a, x^-, v, x^+, b), effectively "stretching" e out to reach v by inserting a narrow triangle $\triangle x^- v x^+$ that sits on e (see Fig. 3b).

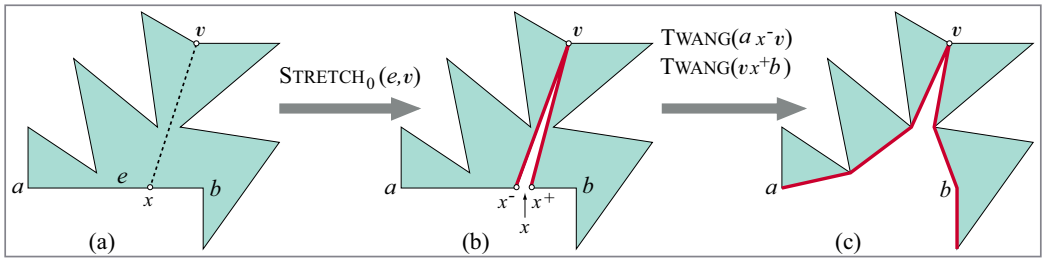


Figure 3: STRETCH(e, v) illustrated (a) v sees $x \in e$ (b) STRETCH$_0(e, v)$ (c) STRETCH(e, v).

To complete the definition of STRETCH(e, v), which removes the pseudovertices x^+ and x^-, we first define the twang operation.

TWANG. Informally, if one views the polygon boundary as an elastic band, a twang operation detaches the boundary from a vertex v and snaps it to v's convex side.

Definition 2.1. *The operation* TWANG(abc) *is defined for any three consecutive vertices* $abc \in \sigma$ *such that*

(1) $\{a, b, c\}$ *are not collinear.*
(2) b *is either a pseudovertex, or a vertex in double contact. If* b *is a vertex in double contact, then* $\triangle abc$ *does not contain a* nested *double contact at* b*. By this we mean the following: Slightly perturb the vertices of* P *to separate each double-contact into two or more points, so that* P *becomes simple. Then* $\triangle abc$ *does not contain any other occurrence of* b *in* σ*. (E.g., in Fig. 4a,* $\triangle a'bc'$ *contains a second occurrence of* b *which prevents snapping* $a'bc'$ *to* b*'s convex side.)*

Under these conditions, the operation TWANG(abc) *replaces the sequence* abc *in* $\mathcal{P}$ *by* sp(abc)*, where* sp(abc) *indicates the shortest path from* a *to* c *that stays inside* $\triangle abc$ *and does not cross* $\partial\mathcal{P}$*. We call* b *the* twang vertex*. Whenever* a *and* c *are irrelevant to the discussion, we denote the twang operation by* TWANG(b)*.*

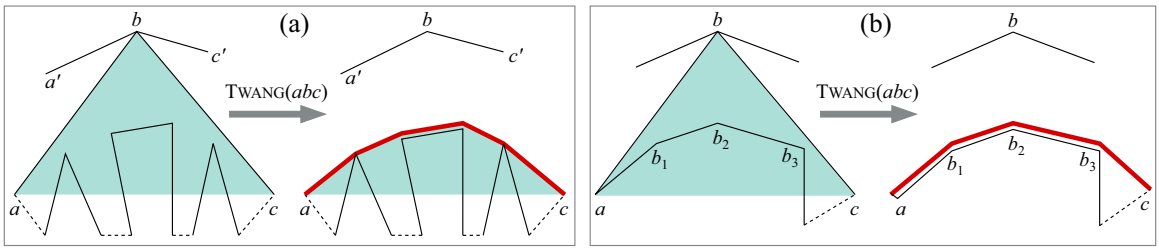


Figure 4: TWANG(abc) illustrated (a) TWANG(abc) replaces abc by sp(abc) (b) TWANG(abc) creates the hairpin vertex a and three doubled edges ab_1, b_1b_2 and b_2b_3.

Informally, TWANG(abc) "snaps" the boundary to wrap around the hull of the points in $\triangle abc$, excluding b (see Fig. 4a). A twang operation can be viewed as taking a step toward simplicity by removing either a pseudovertex or a point of double contact. We should note

that sp(abc) includes every vertex along this path, even collinear vertices. If there are no points inside $\triangle abc$, then sp(abc) = ac, and TWANG(abc) can be viewed as the reverse of HOP(ac, b). If a=c (i.e., ab and bc overlap in $\mathcal{P}$), we call b a *hairpin* vertex of $\mathcal{P}$; in this case, TWANG(aba) replaces aba in $\mathcal{P}$ by a. Hairpin vertices and "doubled edges" arise naturally from twangs. In Fig. 4b for instance, TWANG(abc) produces a hairpin vertex at a and doubled edges ab_1, b_1b_2, b_2b_3. So we must countenance such degeneracies. In general, there are points interior to the triangle, and the twang creates new points of double contact. Below, we will apply twangs repeatedly to remove all double contacts.

STRETCH. We can now complete the definition of STRETCH(e, v), with $e = ab$. First execute STRETCH$_0$(e, v), which picks the two pseudovertices x^+ and x^-. Then execute TWANG(ax^-v) and TWANG(vx^+b), which detach the boundary from x^+ and x^- and return to a polygonal wrap of S (see Fig. 3c). We refer to e (v) as the *stretch edge* (*vertex*).

2.3. Twang Cascades

A twang in general removes one double contact and creates perhaps several others. A TWANGCASCADE applied on a polygonal wrap $\mathcal{P}$ removes all points of double contact from $\mathcal{P}$:

TWANGCASCADE($\mathcal{P}$)

Loop for as long as $\mathcal{P}$ has a point of double contact b:

1. Find a vertex sequence abc in $\mathcal{P}$ that satisfies the twang conditions (cf. Def. 2.1).
2. TWANG(abc).

Note that for any point b of double contact, there always exists a vertex sequence abc that satisfies the twang conditions and therefore the twang cascade loop never gets stuck. That a twang cascade eventually terminates is not immediate. The lemma below shows that TWANG(abc) shortens the perimeter of the polygonal wrap (because it replaces abc by sp(abc)) by at least a constant depending on the geometry of the point set. Therefore, any twang cascade must terminate in a finite number of steps.

Lemma 2.2. *A single twang* TWANG*(abc) decreases the perimeter of the polygonal wrap by at least* $2d_{\min}(1-\sin(\alpha_{\max}/2))$*, where* $d_{\min}$ *is the smallest pairwise point distance and* $\alpha_{\max}$ *is the maximum convex angle formed by any triple of non-collinear points.*

Supplementing this geometric bound, we establish in [DFOR07, App. 3] a combinatorial upper bound of $O(n^n)$ on the number of twangs in any twang cascade. An impediment to establishing a better bound is that a point can twang more than once in a cascade. Indeed we present an example in which $\Omega(n)$ points each twang $\Omega(n)$ times in one cascade, providing an $\Omega(n^2)$ lower bound.

2.3.1. *Forward Move.* We define a *forward move* on a polygonization P of a set S as a stretch (with the additional requirement that the pseudovertices on the stretch edge lie on the reflex side of the stretch vertex), followed by a twang and then a twang cascade, as described below:

FORWARDMOVE(P, e, v)

Preconditions: (i) P is a simple polygon, (ii) e and v satisfy the conditions of STRETCH(e, v), and (iii) v is a noncollinear vertex such that pseudovertices x^+ and x^- on e lie on the reflex side of v. {Let u, v, w be the vertex sequence containing v in P (necessarily unique, since P is simple).}

1. $\mathcal{P} \leftarrow$ STRETCH(e, v).
2. $\mathcal{P} \leftarrow$ TWANG(uvw).
3. $P' \leftarrow$ TWANGCASCADE$(\mathcal{P})$.

A FORWARDMOVE takes one polygonization P to another P' (see Fig. 5), as follows from Lemma 2.2. Note that x^+ and x^- must lie on the reflex side of v (i.e., precondition (iii) of FORWARDMOVE) so that STRETCH(e, v) does not introduce a nested double contact in $\triangle uvw$ which would prevent the subsequent TWANG(uvw). Next we discuss an important phenomenon that can occur during a forward move.

Stretch Vertex Placement. We note that the initial stretch that starts a move might be "undone" by cycling of the cascade. This phenomenon is illustrated in Fig. 5, where the initial STRETCH(ab, v) inserts v between a and b in the polygonal wrap (Fig. 5b), but v ends up between c and b in the final polygonization (Fig. 5f). Thus any attempt to specifically place v in the polygonization sequence between two particular vertices might be canceled by the subsequent cascade. This phenomenon presents a challenge to reducing a polygon to canonical form (discussed in Sec. 5).

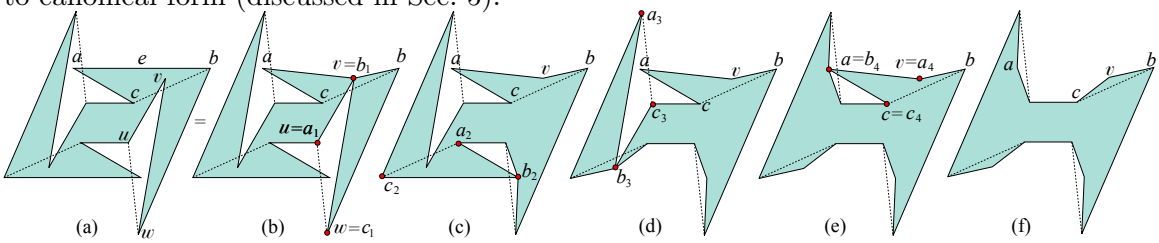


Figure 5: Forward move illustrated. (a) Initial polygon P (b) After STRETCH(ab, v) (c) After TWANG$(a_1b_1c_1)$ (d) After TWANG$(a_2b_2c_2)$ (e) After TWANG$(a_3b_3c_3)$ (f) After TWANG$(a_4b_4c_4)$.

3. Single Pocket Reduction Algorithm

Now that the basic properties of the moves are established, we aim to show that our moves suffice to connect any two polygonizations of a point set S. The plan is to reduce an arbitrary polygonization to the canonical polygonization. En route to explaining this reduction algorithm, we show how to remove any particular pocket by redistributing its vertices to other pockets. This method will be applied repeatedly in Sec. 4 to move all pockets to one particular pocket.

In this section we assume that P has two or more pockets. We use $\mathcal{H}(P)$ to refer to the closed region defined by the convex hull of P, and $\partial\mathcal{H}(P)$ for its boundary. For a fixed hull edge e that is the lid of a pocket A, the goal is to reduce A to e by redistributing the vertices of A among the other pockets, using forward moves only. This is accomplished by the SINGLE POCKET REDUCTION algorithm, which repeatedly picks a hull vertex v of A and attaches v to a pocket other than A; see Fig. 6 for an example run.

SINGLE POCKET REDUCTION(P, e) Algorithm

Loop for as long as the pocket A of P with lid e contains three or more vertices:

1. Pick an edge-vertex pair (e, v) such that
   e is an edge of P on ∂B for some pocket $B \neq A$
   $v \in A$ is a non-lid true corner vertex on $\partial\mathcal{H}(A)$ that sees e
2. $P \leftarrow$ FORWARDMOVE(P, e, v).

We now establish that the SINGLE POCKET REDUCTION algorithm terminates in a finite number of iterations. First we prove a more general lemma showing that a twang operation can potentially reduce, but never expand, the hull of a pocket.

Lemma 3.1 (Hull Nesting under Twangs)**.** *Let A be a pocket of a polygonal wrap $\mathcal{P}$ and let vertex $b \notin \partial\mathcal{H}(\mathcal{P})$ satisfy the twang conditions. Let A' be the pocket with the same lid as A after* TWANG(b). *Then $A' \subseteq \mathcal{H}(A)$.*

Proof: Let abc be the vertex sequence involved in the twang operation. Then TWANG(abc) replaces the path abc by $\mathrm{sp}(abc)$. If abc does not belong to ∂A, then TWANG(abc) does not affect A and therefore $A' \equiv A$. So assume that abc belongs to ∂A. This implies that b is a vertex of A. Note that b is a non-lid vertex, since $b \notin \partial\mathcal{H}(\mathcal{P})$. Then $\triangle abc \subset \mathcal{H}(A)$, and the claim follows from the fact that $\mathrm{sp}(abc) \subset \triangle abc$. □

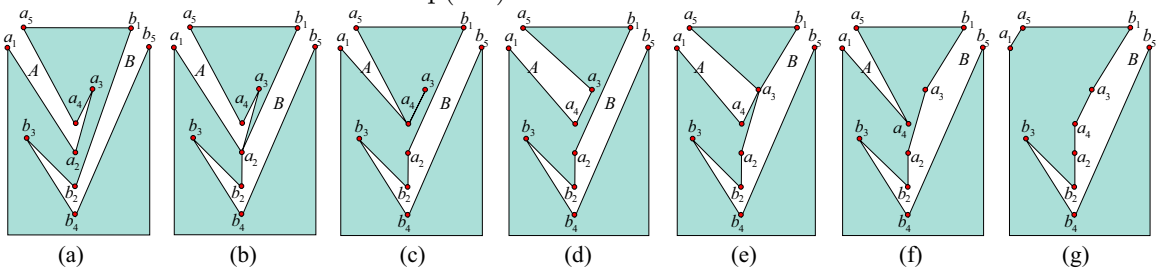


Figure 6: SINGLE POCKET REDUCTION(P, a_1a_5) illustrated: (a) Initial P; (b) After STRETCH(b_1b_2, a_2); (c) After TWANG$(a_1a_2a_3)$; (d) After TWANG$(a_3a_4a_5)$; (e) After STRETCH(a_2b_1, a_3); (f) After TWANG$(a_4a_3a_5)$; (g) After STRETCH(a_2a_3, a_4) + TWANG$(a_1a_4a_5)$.

Lemma 3.2. *The* SINGLE POCKET REDUCTION *algorithm terminates in $O(n)$ forward moves.*

4. Multiple Pocket Reduction Algorithm

For a given hull edge e, the goal is to transform P to a polygon with a single pocket with lid e, using forward moves only. If e is an edge of the polygon, for the purpose of the algorithm discussed here we treat e as a (degenerate) target pocket T. We assume that, in addition to T, P has one or more other pockets, otherwise there is nothing to do. Then we can use the SINGLE POCKET REDUCTION algorithm to eliminate all pockets of P but T, as described in the POCKET REDUCTION algorithm below.

POCKET REDUCTION (P, e) Algorithm

If e is an edge of P, set $T \leftarrow e$, otherwise set $T \leftarrow$ the pocket with lid e
(in either case, we treat T as a pocket).
For each pocket lid $e' \neq e$
Call SINGLE POCKET REDUCTION(P, e')

Observe that the POCKET REDUCTION algorithm terminates in $O(n^2)$ forward moves: there are $O(n)$ pockets each of which gets reduced to its lid edge in $O(n)$ forward moves (cf. Lemma 3.2).

Fig. 7 illustrates the POCKET REDUCTION algorithm on a 17-vertex polygon with three pockets A, B and C, each of which has 3 non-lid vertices, and target pocket T with lid edge $e = t_1t_2$. The algorithm first calls SINGLE POCKET REDUCTION(P, a_1a_5), which transfers to B all non-lid vertices of A, so B ends up with 6 non-lid vertices (this reduction is illustrated in detail in Fig. 6). Similarly, SINGLE POCKET REDUCTION(P, b_1b_5) transfers to C all non-lid vertices of B, so C ends up with 9 non-lid vertices, and finally SINGLE POCKET REDUCTION(P, c_1c_5) transfers all these vertices to T.

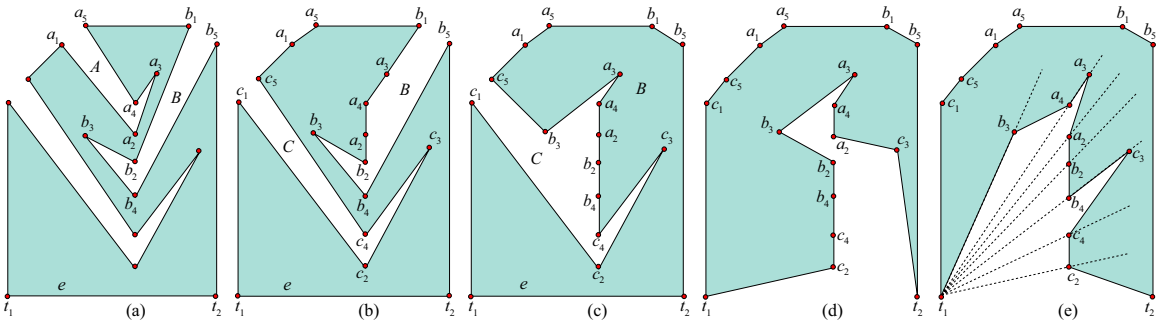


Figure 7: (a-e) POCKET REDUCTION(P, t_1t_2): (a) Initial P; (b) After SINGLE POCKET REDUCTION(P, a_1a_5); (c) After SINGLE POCKET REDUCTION(P, b_1b_5); (d) After SINGLE POCKET REDUCTION(P, c_1c_5); (e) After CANONICAL POLYGONIZATION(P, t_1t_2).

This example shows that the $O(n^2)$ bound on the number of forward moves is tight: an n-vertex polygon with a structure similar to the one in Fig. 7a has $O(n)$ pockets. The number of forward moves performed by the POCKET REDUCTION algorithm is therefore $3 + 6 + 9 + \ldots \frac{3n}{5} = \Theta(n^2)$, so we have the following lemma:

Lemma 4.1. *The* POCKET REDUCTION *algorithm employs* $\Theta(n^2)$ *forward moves.*

5. Single Pocket to Canonical Polygonization

Let $P(e)$ denote an arbitrary one-pocket polygonization of S with pocket lid $e = ab$. Here we give an algorithm to transform $P(e)$ into the canonical polygonization $P_c(e)$. This, along with the algorithms discussed in Secs. 3 and 4, gives us a method to transform any polygonization of S into the canonical form $P_c(e)$. Our canonical polygonization algorithm incrementally arranges pocket vertices in canonical order (cf. Sec. 1.1) along the pocket boundary by applying a series of forward moves to $P(e)$.

CANONICAL POLYGONIZATION(P, e) ALGORITHM

Let $e = ab$. Let $a = v_0, v_1, v_2, \ldots, v_k, v_{k+1} = b$ be the canonical order of the vertices of pocket $P(e)$. For each $i = 1, 2, \ldots, k$

1. Set $\ell_i \leftarrow$ line passing through a and v_i
2. Set $e_{i-1} \leftarrow$ pocket edge $v_{i-1}v_j$, with $j > i - 1$
3. If e_{i-1} is not identical to $v_{i-1}v_i$, apply FORWARDMOVE(e_{i-1}, v_i).

We now show that the one-pocket polygonization resulting after the i-th iteration of the loop above has the points $v_0, \ldots, v_i$ in canonical order along the pocket boundary. (Note that this invariant ensures there is an edge (v_{i-1}, v_j) with $j > i-1$ in Step 2.) This, in turn, is established by showing that the ForwardMove in the i-th iteration involves only points in the set $\{v_i, v_{i+1}, \ldots, v_k\}$. These observations are formalized in the following lemmas [DFOR07, App. 1]:

Lemma 5.1. *The i-th iteration of the* Canonical Polygonization *loop produces a polygonization of S with one pocket with lid e and with vertices $v_0, \ldots, v_i$ consecutive along the pocket boundary.*

Lemma 5.2. *The* Canonical Polygonization *algorithm constructs $P_c(e)$ in $O(n)$ forward moves.*

6. Reverse Moves

Connectivity of the space of polygonizations will follow by reducing two given polygonizations P_1 and P_2 to a common canonical form P_c, and then reversing the moves from P_c to P_2. Although we could just define a reverse move as a time-reversal of a forward move, it must be admitted that such reverse moves are less natural than their forward counterparts. So we concentrate on establishing that reverse moves can be achieved by a sequence of atomic stretches and twangs.

Reverse Stretch. The reverse of Stretch(e, v) may be achieved by a sequence of one or more twangs, as illustrated in Fig. 8a. This result follows from the fact that the "funnel" created by the stretch is empty, and so the twangs reversing the stretch do not cascade.

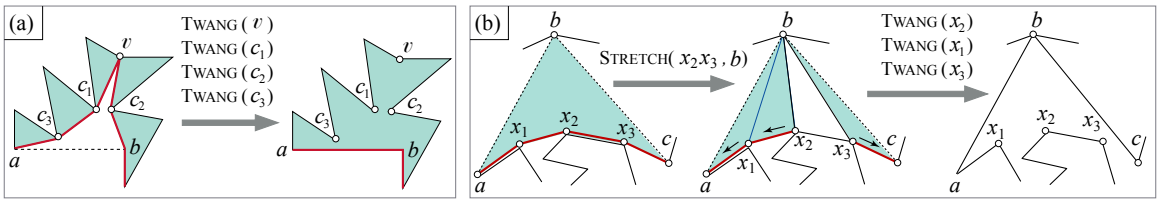


Figure 8: Reverse atomic moves: (a) Stretch(ab, v) is reversed by Twang(v), Twang(c_1), Twang(c_2), Twang(c_3). (b) Twang(b) is reversed by Stretch(x_2x_3, b), Twang(x_2), Twang(x_1) and Twang(x_3).

Reverse Twang. An "untwang" can be accomplished by one stretch followed by a series of twangs. Fig. 8b illustrates how Twang(abc) may be reversed by one Stretch(e, b), for any edge e of sp(abc), followed by zero or more twangs. Observe that the initial stretch in the reverse twang operation is not restricted to the reflex side of the stretch vertex, as it is in a ForwardMove. If b is a hairpin vertex (i.e., a and c coincide), we view ac as an edge of length zero and the reverse of Twang(b) is simply Stretch(e, b).

We have shown that the total effect of any forward move, consisting of one stretch and a twang cascade, can be reversed by a sequence of stretches and twangs. We call this sequence a *reverse move*. One way to view the consequence of the above two results can be expressed via regular expressions. Let the symbols s and t represent a Stretch and Twang respectively. Then a forward move can be represented by the expression st^+: a stretch followed by one or more twangs. A reverse stretch, s^{-1} can be achieved by one or

more twangs: t^+. And a reverse twang t^{-1} can be achieved by st^*. Thus the reverse of the forward move st^+ is $(t^{-1})^+s^{-1} = (st^*)^+t^+$, a sequence of stretches and twangs, at least one of each.

7. Connectivity and Diameter of Polygonization Space

We begin with a summary the algorithm which, given two polygonizations P_1 and P_2 of a fixed point set, transforms P_1 into P_2 using stretches and twangs only.

POLYGON TRANSFORMATION(P_1, P_2) Algorithm

1. Select an arbitrary edge e of $\partial\mathcal{H}(P_1)$.
2. $P_1 \leftarrow$ POCKET REDUCTION(P_1, e); $M_1 \leftarrow$ atomic moves of [$P_2 \leftarrow$ POCKET REDUCTION(P_2, e)].
3. $P_c \leftarrow$ CANONICAL POLYGONIZATION(P_1, e);
   $M_2 \leftarrow$ atomic moves of [CANONICAL POLYGONIZATION(P_2, e).]
4. Reverse the order of the moves in $M_1 \oplus M_2$ ($\oplus$ represents concatenation).
5. For each stretch s (twang t) in $M_1 \oplus M_2$ in order,
   execute reverse stretch s^{-1}(reverse twang t^{-1}) on P_c.

This algorithm, along with Lemmas 4.1 and 5.2, establishes our main theorem:

Theorem 7.1. *The space of polygonizations of a fixed set of n points is connected via a sequence of forward and reverse moves. Each node of the space has degree in $\Omega(n)$ and $O(n^2)$, and the diameter of the polygonization space is $O(n^2)$ moves.*

This diameter bound is tight for our specific algorithm but might not be for other algorithms. Each twang operation can be carried out in $O(n)$ time using a hull routine on the sorted points inside $\triangle abc$; and $\Omega(n)$ might be needed, because sp() might hit $O(n)$ vertices. So the running time of a single forward/reverse move is $T \cdot O(n)$, where T is an upper bound on the number of twangs in a move.

8. Random Polygons

We have implemented a version of random polygon generation. After creating an initial polygonization, we move from polygonization to polygonization via a sequence of forward moves, where additional stretches are permitted in the cascade to simulate reverse moves. Here we report on one experiment that investigates the speed with which the exponential space of polygonizations is explored. We use a variant of the example in Fig. 1a, which has at least 2^k polygonizations. The variant is shown in Fig. 9a, which breaks collinearities by distributing the vertices onto top, middle, and bottom circular arcs. We map each polygonization of this point set to a k-bit binary number, where the k^{th} bit indicates whether the shortest path from the k^{th} middle vertex is to a top (1) or bottom (0) vertex.[4] (Note this map is many-to-one, as there are more than 2^k polygonizations.) Starting from an arbitrary polygonization, we then repeatedly select a random stretch, and twang to quiescence. Figs. 9b,c display the range of the random walk in two formats: (b) shows the number of the 256 bit patterns reached over the 5000 stretches—91% of the patterns were visited by the end of the trial; (c) shows when each bit pattern was reached (dark), with time growing downwards. By the final stretch, 22 patterns (light) were yet to be visited. In this trial, the average length of a twang cascade was 1.2; more precisely, the 5000 stretches invoked 5960 twangs, for a total of 10,960 atomic moves.

[4]Path length is measured by the number of edges, with Euclidean length breaking ties.

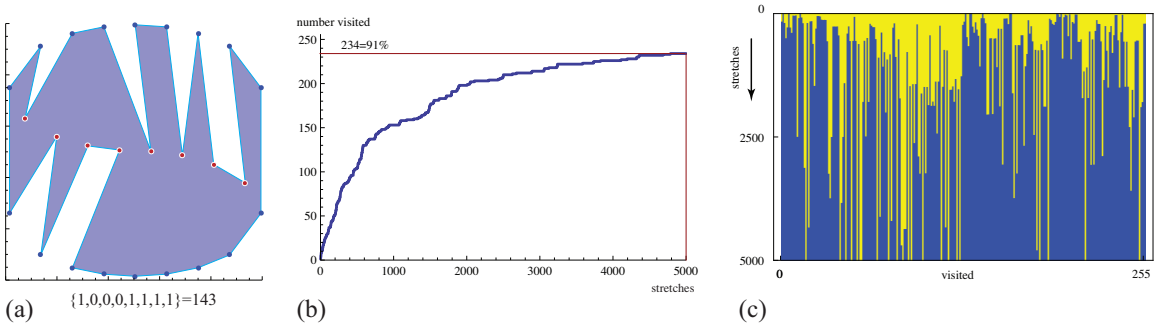


Figure 9: k=8, 2^k=256. (a) Polygonization→bits map. (b) Numbers visited vs. stretches. (c) Dark: numbers visited with increasing stretches; light: not yet visited.

9. Open Problems

Our work leaves many interesting problems open. One unresolved question is whether the number of twangs T in a twang cascade is exponential or if there is a polynomial bound, thereby resolving the computational complexity of the polygon transformation algorithm. We have shown that T is $\Omega(n^2)$ and $O(n^n)$, leaving a large gap to be closed. We would also like to establish a lower bound on the diameter.

In Sec. 7 we established connectivity with forward moves and their reverse, and although both moves are composed of atomic stretches and twangs, the forward moves seem more naturally determined. This suggests the question of whether forward moves suffice to ensure connectivity.

It remains to be seen if the polygonization moves explored in this paper will be effective tools for generating random polygons. One possibility is to start from a doubled random noncrossing spanning tree, which is a polygonal wrap. Finally, we are extending our work to 3D polyhedralizations of a fixed 3D point set.

References

[AH96] T. Auer and M. Held. Heuristics for the generation of random polygons. In *Proc. 8th Canad. Conf. Comput. Geom.*, pages 38–43, 1996.

[CHUZ01] J. Czyzowicz, F. Hurtado, J. Urrutia, and N. Zaguia. On polygons enclosing point sets. *Geombinatorics*, XI (1):21–28, 2001.

[DFOR07] M. Damian, R. Flatland, J. O'Rourke, and S. Ramaswami. Connecting polygonizations via stretches and twangs. `arxiv.org`, arXiv:0709.1942v1 [cs.CG], 2007.

[HHH02] C. Hernando, M. Houle, and F. Hurtado. On local transformation of polygons with visibility properties. *Theoretical Computer Science*, 289(2):919-937, 2002.

[vLS82] J. van Leeuwen and A. A. Schoone. Untangling a travelling salesman tour in the plane. In J. R. Mühlbacher, editor, *Proc. 7th Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, pages 87–98, München, 1982. Hanser.

[ZSSM96] C. Zhu, G. Sundaram, J. Snoeyink, and J. S. B. Mitchell. Generating random polygons with given vertices. *Comput. Geom. Theory Appl.*, 6:277–290, 1996.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 229-240
www.stacs-conf.org

DETERMINISTICALLY ISOLATING A PERFECT MATCHING IN BIPARTITE PLANAR GRAPHS

SAMIR DATTA [1], RAGHAV KULKARNI [2], AND SAMBUDDHA ROY [3]

[1] Chennai Mathematical Institute, Chennai, India.
E-mail address: sdatta@cmi.ac.in

[2] University of Chicago, Chicago, USA.
E-mail address: raghav@cs.uchicago.edu

[3] IBM Research Laboratory, New Delhi, India.
E-mail address: sambuddha@in.ibm.com

Abstract. We present a deterministic way of assigning small (log bit) weights to the edges of a bipartite planar graph so that the minimum weight perfect matching becomes unique. The *isolation lemma* as described in [MVV87] achieves the same for general graphs using a randomized weighting scheme, whereas we can do it deterministically when restricted to bipartite planar graphs. As a consequence, we reduce both decision and construction versions of the matching problem to testing whether a matrix is singular, under the promise that its determinant is 0 or 1, thus obtaining a highly parallel SPL algorithm for bipartite planar graphs. This improves the earlier known bounds of non-uniform SPL by [ARZ99] and NC^2 by [MN95, MV00]. It also rekindles the hope of obtaining a deterministic parallel algorithm for constructing a perfect matching in non-bipartite planar graphs, which has been open for a long time. Our techniques are elementary and simple.

1. Introduction

The *Matching Problem* is one of the most well-studied in the field of parallel complexity. Attempts to solve this problem have led to the development of a variety of combinatorial, algebraic and probabilistic tools which have applications even outside the field. Since the problem is still open, researchers linger around it in search of new techniques, if not to solve it in its whole generality, then at least under various natural restrictions. In this paper, we will focus on the deterministic complexity of the *Matching Problem* under its planar restrictions.

This work was done while the second author was visiting Chennai Mathematical Institute.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

1.1. The Matching Problem

Definition 1.1. A matching in an undirected graph is a collection of edges which have no endpoint in common.

Such a collection of edges is called "independent". See [LP86] for an excellent survey on matchings.

The computational question one can ask here is, given a graph, to find a matching of the maximum cardinality.

Definition 1.2. A perfect matching in a graph is a collection of independent edges which cover all the vertices.

One may ask various computational questions about perfect matchings in graphs. We will consider the following three questions:
Question 1: (Decision) Is there a perfect matching in a given graph ?
Question 2: (Search) Construct a perfect matching in a graph, if it exists.
Question 3: (Uniqueness Testing or UPM) Does a given graph have exactly one perfect matching?
There are polynomial time algorithms for the above graph matching problems and historically people have been interested in studying the parallel complexity of all the three questions above. The UPM question for bipartite graphs is deterministically parallelizable [KVV85] (i.e. it lies in the complexity class NC; see any standard complexity text for a formal definition, say [V99]). Intuitively, NC is a complexity class consisting of the problems having a parallel algorithm which runs in polylogarithmic time using polynomially many processors which have access to a common memory.

It is the class consisting of so called "well parallelizable" problems. NC is inside P - problems having a sequential polynomial time algorithm. Whether the *Matching Problem* is deterministically parallelizable remains a major open question in parallel complexity.

Open Problem 1.3. Is Matching in NC ?

The best we know till now is that *Matching* is in *Randomized* NC. See for example, [KUW86, MVV87]. Several restrictions of the matching problem are known to be in NC, for example, bipartite planar graphs [MN95, MV00], graphs with polynomially bounded number of perfect matchings [GK87] etc. Whether the search version reduces to the decision version has also not been answered yet.

1.2. Randomized Isolation Lemma

Lemma 1.4. [MVV87] *One can randomly assign polynomially bounded weights to the edges of a graph so that with high probability the minimum weight perfect matching becomes unique.*

Using the isolation lemma, [MVV87] obtained a simple *Randomized* NC algorithm for finding a perfect matching in arbitrary graphs.

1.3. Matching in SPL/poly

Allender et al [ARZ99] proved a non-uniform bound for matching problem which allows us to replace the randomization by a polynomial length advice string. Hence, we know that matching is parallelizable with polynomial bit advice.

Definition 1.5. SPL is a promise class that is characterized by the problem of checking whether a a matrix is singular under the promise that its determinant is either 0 or 1. The corresponding non-uniform class SPL/poly is SPL with a polynomial bit advice.

SPL is inside $\oplus$L and inside $\oplus_p$L for all p. While UL (unambiguous Logspace) is inside SPL, NL (nondeterministic Logspace) is incomparable with SPL. Both NL and SPL are known to lie inside NC^2.

Definition 1.6. A language is said to be in SPL/poly if for every positive integer n there exists an advice string A_n such that:

- length of A_n is polynomially bounded in n
- once A_n is given, the membership of any input of size n can be decided in SPL.

Theorem 1.7. [ARZ99] *Matching is in* SPL*/poly.*

1.4. Matchings in Planar Graphs and Deterministic Isolation

The situation for planar graphs is interesting because of the fact that counting the number of perfect matchings in planar graph is in NC ([K67, V88]) whereas constructing one perfect matching is not yet known to be parallelizable. However, for bipartite planar graphs, people have found NC algorithms [MN95, MV00].

The isolation lemma crucially uses randomness in order to isolate a minimum weight set in an arbitrary set system. It is conceivable, however, to exploit some additional structure in the set system to eliminate this randomness. Indeed, recently [BTV07] building upon a technique from [ADR05] were able to isolate a directed path in a planar graph by assigning small *deterministic* weights to the edges. The lemma that sits at the heart of that result says that there is a simple deterministic way to assign weights so that each directed cycle (in a grid graph) gets a non-zero weight. This is shown to imply that if two paths get the same weight neither of them is a min-weight path.

Motivated by their result we explore the possibility of such an isolation for perfect matchings in planar graphs. Our attempt is to assign weights so that the alternating sum is non-zero for each alternating cycle - here alternating sum is the signed sum of weights where the sign is opposite for successive edges. Since alternating cycle result from the super-imposition of two matchings, we are able to isolate a min-weight matching.

Therefore, we are able to devise an NC algorithm for bipartite planar graphs which is conceptually simple, different from the other known algorithms and tightens its complexity to the smaller class SPL. The search problem for matching in non-bipartite planar graphs still remains open even though the corresponding decision and counting versions are known to be in NC. Our algorithm rekindles the hope for solving general planar matching in NC.

2. Preliminaries

Here we describe the technical tools that we need in the rest of the paper. Refer to any standard text (e.g. [V99]) for definitions of the complexity classes $\oplus$L, $\oplus_p$L, NL, UL, NC^2 . For graph-theoretic concepts, for instance, *planar graph*, *outerplanar graph*, *spanning trees*, *adjacency matrix*, *Laplacian matrix* of a graph, we refer the reader to any standard text in graph theory (e.g. [D05]).

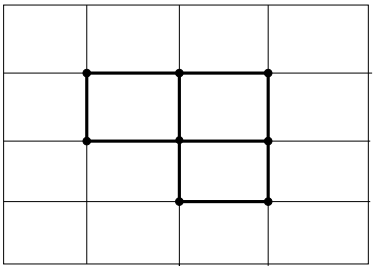


Figure 1: A Grid Graph

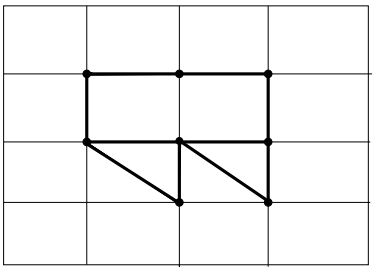


Figure 2: An Almost Grid Graph

2.1. Definitions and Facts

We will view an $n \times n$ grid as a graph simply by putting the nodes at the grid points and letting the grid edges act as the edges of the graph.

Definition 2.1. *Grid graphs* are simply subgraphs of an $n \times n$ grid for some n. See Figure 1 for an example. We call each unit square of the grid a *block*.

Definition 2.2. We call a graph an *almost grid graph* if it consists of a grid graph and possibly some additional diagonal edges which all lie in some single row of the grid. Moreover all the diagonal edges are parallel to each other. See Figure 2.

In this paper we will consider weighted grid graphs where each edge is assigned an integer weight.

Definition 2.3. (1) Given a grid, assign a "+" sign to all the vertical edges and a "-" sign to all the horizontal edges.
(2) Assign a sign of $(-1)^{i+j}$ to the block in the i^{th} row and j^{th} column (adjacent blocks get opposite signs).

Definition 2.4. Given a weighted grid graph G, the *circulation* of a block B(denoted $circ(B)$) in G is the signed sum of weights of the edges of it: $circ(B) = \sum_{e \in B} sign(e) weight(e)$.

Definition 2.5. Given a weighted grid graph G and a simple cycle $C = (e_0, e_1, \ldots, e_{2k-1})$ in it, where e_0 is, say, the leftmost topmost vertical edge of C; we define the circulation of a cycle C as $circ(C) = \sum_{i=0}^{2k-1} (-1)^i weight(e_i)$.

The following lemma plays a crucial role in constructing non-vanishing circulations in grid graphs as will be described in the next section.

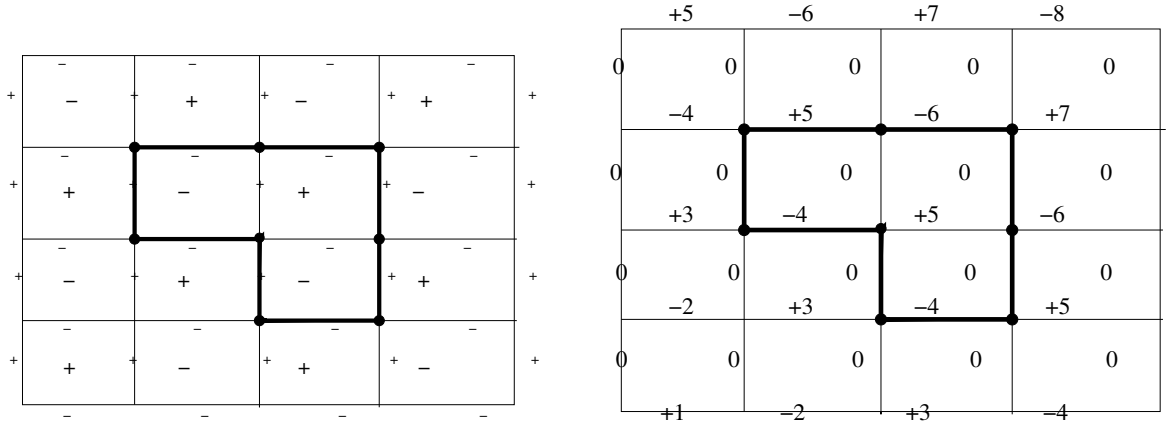


Figure 3: Signs and Weights of the blocks and the edges of a grid

Lemma 2.6. Block Decomposition of Circulations: *The absolute value of the circulation of a simple cycle C in a grid graph G is equal to the signed sum of the circulations of the blocks of the grid which lie in the interior of C.*
$|circ(C)| = \sum_{B \in interior(C)} sign(B) circ(B)$

Proof. Consider the summation on the right hand side. The weight of any edge in the *interior* of C will get cancelled in the summation because that edge will occur in exactly two blocks which are adjacent and hence with opposite signs. Now what remains are the boundary edges. Call two boundary edges *adjacent* if they appear consecutively on the cycle C.

Claim 2.7. *Adjacent boundary edges get opposite signs in the summation on the right hand side above.*

Proof. We have to consider two cases, either the adjacent boundary edges lie on adjacent blocks, in which case since adjacent blocks have opposite signs, these edges will also get opposite signs as they are both vertical or horizontal edges. See Figure 3. In the other case, when adjacent boundary edges do not lie on adjacent blocks, they lie on two blocks which are diagonally next to each other. In this case, both blocks will have the same sign but since one edge is vertical and the other is horizontal, the effective sign of the edges will be opposite. See Figure 3. Hence, the adjacent boundary edges will get opposite sign in the summation. This completes the proof that the right hand side summation is precisely + $circ(C)$ or - $circ(C)$. ■

■

We will also have occasion to employ the following lemma and we record it here:

Lemma 2.8. Temperley's Bijection: *The spanning trees of a planar graph are in one to one correspondence with perfect matchings in a bipartite planar graph. Moreover the correspondence is weight preserving.*

This bijection was first observed by Temperley around 1967. Recently [KPW00] have found a *Generalized Temperley Bijection* which gives a one-to-one weight preserving mapping between directed rooted spanning trees or arborescences in a directed planar graph and perfect matchings in an associated bipartite planar graph.

2.2. Planar Matching and Grid Graphs

Grid graphs have turned out to be useful for solving the reachability question in directed planar graphs, cf. [ABCDR06, BTV07]. Motivated by this fact we explore the possibility of reducing planar matching problem to that of grid graphs. Non-bipartiteness becomes an obstacle here which leaves us with the following observations:

Lemma 2.9. *One can convert any bipartite planar graph into a grid graph such that the perfect matchings remain in one-to-one correspondence.*

Proof. This is described in [DKLM07]. It follows closely the procedure for embedding a planar graph into a grid, described by [ABCDR06]. ∎

Though non-bipartiteness is an issue, we can get rid of it to a certain extent, though as expected, not completely .

Lemma 2.10. *Any planar graph, not necessarily bipartite, can be converted to an* almost grid graph *while maintaining the one to one correspondence between the perfect matchings.*

Proof. This procedure is analogous to the previous one except that we can observe that the edges which are causing non bipartiteness can be elongated into a long path and placed in a grid so that only in a single row one needs to use a diagonal edge. ∎

3. Bipartite Planar Perfect Matching in SPL

In this section, we will give a simple algorithm for finding a perfect matching in bipartite planar graphs, also improving over its complexity by putting it in SPL. Earlier the best known bound was NC^2. See for example [MN95, MV00]. At the core of our algorithm, lies a procedure to deterministically assign the small (logarithmic bit long) weights to the edges of a bipartite planar graph, so that the minimum weight perfect matching becomes unique. A simple observation about non-vanishing circulations in bipartite planar graphs makes it possible to isolate a perfect matching in the graph, which can be further extracted out using an SPL query.

3.1. Non-vanishing Circulations in Grid Graphs

We are interested in assigning the small weights to the edges of a grid so that any cycle in it will have non-zero circulation. This weighting scheme is at the heart of the isolation of perfect matchings in grid graphs. The procedure runs in Logspace.

Lemma 3.1. *One can assign, in Logspace, small (logarithmic bit) weights to the edges of a grid so that circulation of any cycle becomes non-zero. (One weighting scheme which guarantees non-zero circulation for every cycle in the grid is shown in the Figure 3.)*

Proof. We assign all vertical edges weight 0 and horizontal edge from grid point (i, j) to $(i+1, j)$ is assigned a weight of $(-1)^{i+j}(i+j+1)$ as shown in figure 3. Thus the circulation of the block with diagonally opposite vertices (i, j) and $(i+1, j+1)$ is $\sum_e sign(e)weight(e)$ $= (-1)(-1)^{i+j}(i+j+1) + (+1)0 + (-1)(-1)^{i+j+1}(i+j+2) + (+1)0 = (-1)^{i+j}$

Thus, the weighting scheme makes sure that the circulation of any block is either +1 or - 1. Moreover, the circulation of a block is positive if and only if its sign is positive. Now,

using the Block Decomposition of Circulations (Lemma 2.6), we have that the circulation of any cycle in absolute value is precisely the number of blocks in the interior of it, and hence is never zero. ∎

3.2. Non-vanishing Circulations: A Direct Method

One can think of the procedure of assigning the weights to the edges of bipartite planar graph without having to convert it to a grid graph. The procedure is as follows:

- **Step 1:** *Make the graph Eulerian (every vertex has an even degree):* add spurious edges to it without disturbing the bipartiteness.
  - *Step 1.1:* Perform an Euler traversal on a spanning tree in the dual graph.
  - *Step 1.2:* While performing the traversal, make sure that when you leave the face, all the vertices in the face, except for the end points of the edge through which we go to the next face, are of even degree. To guarantee this we can do the following:
    - *Step 1.2 a) :* Start with one end point say u of the edge (u, v) through which we go to the next face. Visit all the vertices of the face in a cyclic ordering, every time connect an odd degree vertex to the next vertex by a spurious edge.
    - *Step 1.2 b) :* If the next vertex is also of odd degree then go to its next vertex. If the next vertex is of even degree then we have pushed the oddness one step further.
    - *Step 1.2 c) :* Continue the same procedure till we remove all the oddness or push it to v.
    - *Step 1.2 d) :* In the process, the graph might become a multi-graph i.e. between two nodes we may have multiple edges, but this can be taken care of by replacing every multi-edge by a path of length 3. The bipartiteness is preserved in the process.
- **Step 2:***Fix the signs:* After Step 1, the graph has become Eulerian, and hence the dual graph is bipartite.
  - *Step 2 a) Assign alternating signs to adjacent faces:* Form a bipartition of the dual, and fix all the faces in one bipartition to have + sign and the others to have - sign. Any two adjacent faces will have opposite signs. Here, faces will act as blocks.
  - *Step 2 b) Assign alternating signs to adjacent edges of every face:* Consider an auxiliary graph obtained from the bipartite planar graph as follows: Every new vertex corresponds to an edge in the graph. Join two new vertices by a new edge iff the corresponding edges in the original graph are adjacent along some face. Now since both the original graph and its dual are bipartite , the auxiliary graph will also be bipartite - hence edges in the two bipartitions get opposite signs ensuring that around every face the signs are alternating.
- **Step 3:** *Assign small weights to the edges:* Now make another Euler traversal on the dual tree everytime assigning the weight to the dual tree edge through which you traverse to the next face so that the circulation of the face you leave is exactly same as the sign of the face. All non-tree edges will be assigned zero weight. It is easy to see that all the weights assigned are small (logarithmic bit).

Block Decomposition of Circulations: Again, the circulation of a cycle will decompose into signed sum of circulations of the faces in the interior of it and since the sign and the circulation for any face is the same, we will have non-vanishing circulations in the graph. The details are analogous to the case of a grid. We leave the details to the reader.

3.3. Deterministic Isolation

The non-vanishing circulations immediately give us the isolation for the perfect matchings.

Lemma 3.2. *If all the cycles in a bipartite graph have non-zero circulations, then the minimum weight perfect matching in it is unique.*

Proof. If not, then we have two minimum weight perfect matchings M_1 and M_2 which will contain some alternating cycles, and each such cycle is of even length. Consider any one such cycle. Since the circulation of an even cycle is nonzero either the part of it which is in M_1 is lighter or the part of it which is in M_2 is lighter, in either case, we can form another perfect matching which is lighter than the minimum weight perfect matching, which is a contradiction. ∎

Thus we have a deterministic way of isolating a perfect matching in bipartite planar graphs, and it is easy to check that the procedure of assigning the weights to the edges works in deterministic Logspace.

3.4. Extracting the Unique Matching

Once we have isolated a perfect matching, one can extract it out easily as follows:

- **Step 1:** Construct an $n \times n$ matrix M associated with a planar graph on n vertices as follows: Find a *Pfaffian orientation* ([K67]) of the planar graph and put the (i, j)th entry of the matrix M to be $x^{w_{(i,j)}}$ where x is a variable and $w_{(i,j)}$ is the weight of the edge (i, j) which is directed from i to j in the Pfaffian orientation. If the edge is directed from j to i then put $-x^{w_{(i,j)}}$ as $(i, j)th$ entry of the matrix.
- **Step 2:** If t is the weight of the minimum weight perfect matching, then the coefficient of x^{2t} in determinant of M will be the number of perfect matchings of weight t. Now, as shown in [MV97, V99] this coefficient can be written as a determinant of another matrix.
- **Step 3:** Now start querying from $i = -n^2$ to $+n^2$ whether the coefficient of x^{2i} is zero or not and the first time you find that it is nonzero; stop. The first nonzero value will occur when $i = t$ and it will be 1 since the minimum weight perfect matching is unique. Hence, during the process, every time we have a promise that if the determinant is non-zero, it is 1. This procedure gives the weight of the minimum weight perfect matching.
- **Step 4:** Now once we know the procedure to find the weight of the minimum weight perfect matching, then one can find out which edges are in the matching by simply deleting the edge and again finding the weight of the minimum weight perfect matching in the remaining graph. If the edge is in the the isolated minimum weight perfect matching then after its deletion the weight of the new minimum weight perfect matching will increase. Otherwise we can conclude that the edge is

not in the isolated minimum weight perfect matching. Note that the isolation holds even after deleting an edge from the graph.

Theorem 3.3. *Bipartite Planar Perfect Matching is in* SPL.

Proof. The Logspace procedure in Lemma 3.1 assigns the small weights to the edges of the graph so that the minimum weight perfect matching is unique and the above procedure extracts it out in $L^{\mathsf{SPL}} = \mathsf{SPL}$. ∎

We obtain the following corollaries.

Corollary 3.4. UPM *in bipartite planar graphs is in* SPL.

Proof. To check whether the graph has unique perfect matching, one can construct one perfect matching and check that after removing any edge of it there is no other perfect matching. ∎

Corollary 3.5. *Minimum weight perfect matching in planar graphs with polynomially bounded weights is computable in* SPL.

Proof. One can first scale the polynomially bounded weights by some large multiplicative factor, say n^4 and then perturb them using the weighting scheme described above so that some minimum weight matching with original weights remains minimum weight matching with new weights and is unique. Then extraction can be done in SPL. ∎

Corollary 3.6. *Minimum weight spanning tree in planar graphs is computable in* SPL *if the weights are polynomially bounded. (The same is true for directed rooted spanning trees (arborescences) in planar graphs due to Generalized Temperley's bijection shown in* [KPW00]*.)*

Proof. This follows from Temperley's bijection. ∎

Restricting the family of planar graphs, yields better upper bounds for Matching questions. Notably, we prove that:

Corollary 3.7. *(of Theorem 3.3) Perfect Matching in outerplanar graphs is in* SPL.

Proof. If we have an outerplanar(1-page) graph on n vertices with vertices labelled from 1 to n along the spine, then observe that the edges between two odd labelled vertices can not be part of any perfect matching. This is because the number of vertices below that edge is odd. Similarly edges between two even labelled vertices can not participate in any perfect matching. Hence, by removing such edges we can make the graph bipartite and then we can apply the previous theorem. ∎

We use the lemma below in order to prove the theorem that follows it.

Lemma 3.8. *The parity of the number of perfect matchings in an outerplanar graph can be computed in Logspace.*

Proof. It is not hard to observe that the parity of the determinant of the adjacency matrix of a graph is the same as the parity of number of perfect matchings in it. Finding the parity of the adjacency matrix of an outerplanar graph can be reduced to finding the parity of the number of spanning trees in an auxiliary planar graph which is constructed by adding a new vertex and connecting it to all the odd degree vertices of the original graph. The new graph has all the vertices of even degree. Hence the adjacency matrix of the new graph is

the same as its Laplacian modulo 2. Now the minor obtained by removing the row and the column corresponding to the new vertex, is precisely the adjacency matrix of the original outerplanar graph modulo 2. Also the determinant(mod 2) of this minor is precisely the parity of the spanning trees in new graph. As shown in [BKR07], the parity of spanning trees modulo 2 in planar graphs can be computed in Logspace. Hence, the parity of the determinant of the adjacency matrix of an outerplanar graph can be obtained in Logspace which in turn gives the parity of the number of perfect matchings in it. ∎

Theorem 3.9. UPM *in outerplanar graphs is in Logspace.*

Proof. If the perfect matching in an outerplanar graph is unique, one can obtain one perfect matching in Logspace. For every edge, one just needs to compute the parity of the number of perfect matchings in the graph with that particular edge removed. If this parity is 1 then don't include this edge in the perfect matching, otherwise do. Now we just need to verify that the perfect matching thus constructed is unique. As seen in Corollary 3.7 we can assume that the outerplanar graph is bipartite. Now, if we consider an auxiliary directed graph obtained from this outerplanar graph by putting a directed edge starting from a vertex and ending in another vertex after following a matching edge starting at the vertex and then a non-matching edge from there, then, this auxiliary graph will have a directed cycle if and only if the matching we start with is not unique. It is possible to show that the auxiliary graph is outerplanar. Finally, since the reachability in directed outerplanar graphs is in Logspace ([ABCDR06]), we have that UPM in outerplanar graphs is in Logspace. ∎

4. Discussion

We saw in Section 3.3 that a perfect matching in bipartite planar graphs can be isolated by assigning small weights to the edges. In this section we discuss the possibility of generalizing this result in two orthogonal directions. For non-bipartite planar graphs and for bipartite but non-planar graphs. The motivation is to isolate a perfect matching in a graph by having non-zero circulation on it.

4.1. Non-bipartite Planar Matching

Though non-bipartiteness is an issue, we can get rid of it to a certain extent, though as expected, not completely .

Lemma 4.1. *Perfect Matching problem in general planar graphs reduces to that of almost grid graphs.*

Now it suffices to get a non-vanishing circulations in almost grid graphs to solve planar matching question. Unfortunately we don't yet know any way of achieving this though we have some observations which might be helpful.

Lemma 4.2. *One can have non-zero circulations for all the even cycles in the graph in the Figure 4. (The graph is simply one row of the grid with diagonals.)*

Proof. Observe that any even cycle in such a graph will either fall in the grid or will fall in the grid formed by horizontal edge together with diagonal edges. Now, its easy to assign the weights as shown in the Figure 4 so that all the horizontal edges get weight 0 while vertical and diagonal edges get weights so that any cycle in vertical or diagonal grid has non-vanishing circulation. ∎

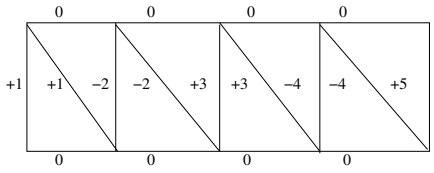


Figure 4: Non-vanishing circulation in a non-bipartite graph

In summary, non-bipartiteness seems to be an issue which is difficult to get around. Hence, we keep the bipartiteness and next we explore the possibility of generalizing our result for non-planar graphs.

4.2. Bipartite Non-planar Matching

Instead of looking at two dimensional grid we now consider three dimensional grids. The matching problem remains as hard as that for general bipartite graphs in this restriction as well.

Lemma 4.3. *One can embed any bipartite graph in a three dimensional grid while preserving matchings.*

Proof. Firstly, one can make the degree of the graph bounded by 3. Then one can use the third dimension to make the space for crossings in the graph. ∎

Open Problem 4.4. Is the perfect matching problem for subgraphs of a three dimensional grid of height 2 (constant height in general) in NC ?

The deterministic isolation of perfect matching is possible through non-vanishing circulations as seen in Section 3.3.

Open Problem 4.5. Is small (log bit) weight non-vanishing circulation possible in every bipartite graph?

4.3. Other Variations

We know that the reachability in directed planar graphs reduces to bipartite planar matching while the reachability in layered grid graphs reduces to the UPM question in the same [DKLM07]. Note that the isolation step in our algorithm works in Logspace. Solving the perfect matching question in bipartite planar graphs in Logspace might be too strong to expect but at least one can ask the question about UPM which would put layered grid graph reachability in Logspace or giving an orthogonal bound for the same.

Open Problem 4.6. Is bipartite planar UPM in L?

We saw how to isolate a perfect matching in bipartite planar graph. The isolation holds for maximum matching in bipartite planar graphs. However, we do not know how to extract out the maximum weight perfect matching in NC.

Open Problem 4.7. Is it possible to extract out the isolated maximum matching in NC even for bipartite planar graphs?

Finally, the question of constructing a perfect matching in planar graphs in NC still remains open.

5. Acknowledgements

We thank Meena Mahajan for useful discussion and comments on the preprint.

References

[ABCDR06] Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta, Sambuddha Roy: Grid Graph Reachability Problems. IEEE Conference on Computational Complexity 2006: 299-313

[ADR05] Eric Allender, Samir Datta, Sambuddha Roy: The Directed Planar Reachability Problem. FSTTCS 2005: 238-249

[ARZ99] Eric Allender, Klaus Reinhardt, Shiyu Zhou: Isolation, Matching, and Counting: Uniform and Nonuniform Upper Bounds. J. Comput. Syst. Sci. 59(2): 164-181 (1999)

[BKR07] Mark Braverman, Raghav Kulkarni, Sambuddha Roy: Parity Problems in Planar Graphs. IEEE Conference on Computational Complexity 2007: 222-235

[BTV07] Chris Bourke, Raghunath Tewari, N. V. Vinodchandran: Directed Planar Reachability is in Unambiguous Log-Space. IEEE Conference on Computational Complexity 2007: 217-221

[DKLM07] Samir Datta, Raghav Kulkarni, Nutan Limaye, Meena Mahajan: Planarity, Determinants, Permanents, and (Unique) Matchings. CSR 2007: 115-126

[D05] Reinhard Diestel. Graph Theory. *Springer*, 2005

[GK87] D. Grigoriev and M. Karpinski. The matching problem for bipartite graphs with polynomially bounded permanent is in NC. In *Proceedings of 28th IEEE Conference on Foundations of Computer Science, pages 166-172.* IEEE Computer Society Press, 1987.

[K67] P. W. Kasteleyn. Graph theory and crystal physics. In F. Harary, editor, *Graph Theory and Theoretical Physics*, page 43-110, Academic Press, 1967.

[KPW00] Richard W. Kenyon, James G. Propp, David B. Wilson, Trees and Matchings, Electronic Journal of Combinatorics, 7(1)

[KUW86] Richard Karp, Eli Upfal, Avi Wigderson. Constructing a perfect matching is in random NC. *Combinatorica, 6:35-48*, 1986.

[KVV85] Dexter Kozen, Umesh V. Vazirani, Vijay V. Vazirani. NC Algorithms for Comparability Graphs, Interval Graphs, and Testing for Unique Perfect Matching. FSTTCS 1985: 496-503

[LP86] L. Lovasz, M. Plummer, Matching Theory, North-Holland, 1986.

[MN95] Gary Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal of Computing*, 24:1002-1017, 1995.

[MV97] Meena Mahajan, V. Vinay: A Combinatorial Algorithm for the Determinant. SODA 1997: 730-738

[MV00] Meena Mahajan, Kasturi Varadarajan. A new NC algorithm to find a perfect matching in planar and bounded genus graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing (STOC)*, pages 351-357, 2000.

[MVV87] Ketan Mulmuley, Umesh Vazirani, Vijay Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1): 105-131, 1987.

[V88] Vijay Vazirani. NC Algorithms for Computing the Number of Perfect Matchings in $K_{3,3}$–free Graphs and Related Problems. SWAT 1988: 233-242

[V99] Heribert Vollmer, Introduction to Circuit Complexity - A Uniform Approach; Texts in Theoretical Computer Science. An EATCS Series. Springer Verlag, 1999.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 241-252
www.stacs-conf.org

TIGHT BOUNDS FOR BLIND SEARCH ON THE INTEGERS

MARTIN DIETZFELBINGER [1], JONATHAN E. ROWE [2], INGO WEGENER [3], AND PHILIPP WOELFEL [4]

[1] Fakultät für Informatik und Automatisierung, Technische Univ. Ilmenau, 98684 Ilmenau, Germany
E-mail address: martin.dietzfelbinger@tu-ilmenau.de

[2] School of Computer Science, University of Birmingham, Birmingham B15 2TT, United Kingdom
E-mail address: J.E.Rowe@cs.bham.ac.uk

[3] FB Informatik, LS2, Universität Dortmund, 44221 Dortmund, Germany
E-mail address: ingo.wegener@uni-dortmund.de

[4] Department of Computer Science, University of Calgary, Calgary, Alberta T2N 1N4, Canada
E-mail address: woelfel@cpsc.ucalgary.ca

Abstract. We analyze a simple random process in which a token is moved in the interval $A = \{0, \ldots, n\}$: Fix a probability distribution μ over $\{1, \ldots, n\}$. Initially, the token is placed in a random position in A. In round t, a random value d is chosen according to μ. If the token is in position $a \geq d$, then it is moved to position $a - d$. Otherwise it stays put. Let T be the number of rounds until the token reaches position 0. We show tight bounds for the expectation of T for the optimal distribution μ. More precisely, we show that $\min_\mu\{E_\mu(T)\} = \Theta\left((\log n)^2\right)$. For the proof, a novel potential function argument is introduced. The research is motivated by the problem of approximating the minimum of a continuous function over $[0, 1]$ with a "blind" optimization strategy.

1. Introduction

For a positive integer n, assume a probability distribution μ on $X = \{1, \ldots, n\}$ is given. Consider the following random process. A token moves in $A = \{0, \ldots, n\}$, as follows:

- Initially, place the token in some position in A.
- In round t: The token is at position $a \in A$. Choose an element d from X at random, according to μ. If $d \leq a$, move the token to position $a - d$ (the step is "accepted"), otherwise leave it where it is (the step is "rejected").

Work of the first author was done in part while visiting ETH Zürich, Switzerland. The third author was supported in part by the DFG collaborative research project SFB 531. Work of the last author was done in part while at the University of Toronto, supported by DFG grant WO1232/1-1 and by SUN Microsystems. Joint work on this topic was initiated during Dagstuhl Seminar 06111 on Complexity of Boolean Functions (2006).

When the token has reached position 0, no further moves are possible, and we regard the process as finished.

At the beginning the token is placed at a position chosen uniformly at random from $\{1,\ldots,n\} = A - \{0\}$. (For simplicity of notation, we prefer this initial distribution over the possibly more natural uniform distribution on $\{0,\ldots,n\}$. Of course, there is no real difference between the two starting conditions.) Let T be the number of rounds needed until position 0 is reached. A basic performance parameter for the process is $\mathbf{E}_\mu(T)$. As μ varies, the value $\mathbf{E}_\mu(T)$ will vary. The probability distribution μ may be regarded as a strategy. We ask: How should μ be chosen so that $\mathbf{E}_\mu(T)$ is as small as possible?

It is easy to exhibit distributions μ such that $\mathbf{E}_\mu(T) = O((\log n)^2)$. (All asymptotic notation in this paper refers to $n \to \infty$.) In particular, we will see that the "harmonic distribution" given by

$$\mu_{\mathrm{har}}(d) = \frac{1}{d \cdot H_n}\,, \text{ for } 1 \le d \le n, \tag{1.1}$$

where $H_n = \sum_{1\le d\le n} \frac{1}{d}$ is the nth harmonic number, satisfies $\mathbf{E}_{\mu_{\mathrm{har}}}(T) = O((\log n)^2)$. As the main result of the paper, we will show that this upper bound is optimal up to constant factors: $\mathbf{E}_\mu(T) = \Omega((\log n)^2)$, for every distribution μ. For the proof of this lower bound, we introduce a novel potential function technique, which may be useful in other contexts.

1.1. Motivation and Background: Blind Optimization Strategies

Consider the problem of minimizing a function $f : [0,1] \to \mathbb{R}$, in which the definition of f is unknown: the only information we can gain about f is through trying sample points. This is an instance of a *black box optimization problem* [1]. One algorithmic approach to such problems is to start with an initial random point, and iteratively attempt to improve it by making random perturbations. That is, if the current point is $x \in [0,1]$, then we choose some distance $d \in [0,1]$ according to some probability distribution μ on $[0,1]$, and move to $x + d$ or $x - d$ if this is an improvement. The distribution μ may be regarded as a "search strategy". Such a search is "blind" in the sense that it does not try to estimate how close to the minimum it is and to adapt the distribution μ accordingly. The problem is how to specify μ. Of course, an optimal distribution μ depends on details of the function f.

The difficulty the search algorithm faces is that for general functions f there is no information about the scale of perturbations which are necessary to get close to the minimum. This leads us to the idea that the distribution might be chosen so that it is *scale invariant*, meaning that steps of all "orders of magnitude" occur with about the same probability. Such a distribution is described in [4]. One starts by specifying a minimum perturbation size ε. Then one chooses the probability density function $h(t) = 1/(pt)$ for $\varepsilon \le t \le 1$, and $h(t) = 0$ otherwise, where $p = \ln(1/\varepsilon)$ is the *precision* of the algorithm. (A random number distributed according to this density function may be generated by taking $d = \exp(-pu)$, where u is uniformly random in $[0,1]$.)

For general functions f, no analysis of this search strategy is known, but in experiments on standard benchmark functions it (or higher dimensional variants) exhibits a good performance. (For details see [4].) From here on, we focus on the simple case where f is *unimodal*, meaning that it is strictly decreasing in $[0, x_0]$ and strictly increasing in $[x_0, 1]$, where x_0 is the unknown minimum point.

Remark 1.1. If one is given the information that f is unimodal, one will use other, deterministic search strategies, which approximate the optimum up to ε within $O(\log(1/\varepsilon))$

steps. As early as 1953, in [3], "Fibonacci search" was proposed and analyzed, which for a given tolerance ε uses the optimal number of steps in a very strong sense.

The "blind search" strategy from [4] can be applied to more general functions f, but the following analysis is valid only for unimodal functions. If the distance of the current point x from the optimum x_0 is $\tau \geq 2\varepsilon$ then every distance d with $\frac{\tau}{2} \leq d \leq \tau$ will lead to a new point with distance at most $\tau/2$. Thus, the probability of at least halving the distance to x_0 in one step is at least $\frac{1}{2}\int_{\tau/2}^{\tau} \frac{dt}{pt} = \frac{\ln 2}{2p}$, which is independent of the current state x. Obviously, then, the expected number of steps before the distance to x_0 has been halved is $2p/\ln 2$. We regard the algorithm to be successful if the current point has distance smaller than 2ε from x_0. To reach this goal, the initial distance has to be halved at most $\log(1/\varepsilon)$ times, leading to a bound of $O(\log(1/\varepsilon)^2)$ for the expected number of steps.

The question then arises whether this is the best that can be achieved. Is there perhaps a choice for μ that works even better on unimodal functions? To investigate this question, we consider a discrete version of the situation. The domain of f is $A = \{0, \ldots, n\}$, and f is strictly increasing, so that f takes its minimum at $x_0 = 0$. In this case, the search process is very simple: the actual values of f are irrelevant; going from a to $a+d$ is never an improvement. Actually, the search process is fully described by the simple random process from Section 1. How long does it take to reach the optimal point 0, for a μ chosen as cleverly as possible? For $\mu = \mu_{\mathrm{har}}$, we will show an upper bound of $O((\log n)^2)$, with an argument very similar to that one leading to the bound $O(\log(1/\varepsilon)^2)$ in the continuous case. The main result of this paper is that the bound for the discrete case is optimal.

1.2. Formalization as a Markov chain

For the sake of simplicity, we let from now on $[a, b]$ denote the discrete interval $\{a, \ldots, b\}$ if a and b are integers. Given a probability distribution μ on $[1, n]$, the Markov chain $R = (R_0, R_1, \ldots)$ is defined over the state space $A = [0, n]$ by the transition probabilities

$$p_{a,a'} = \begin{cases} \mu(a - a') & \text{for } a' < a; \\ 1 - \sum_{1 \leq d \leq a} \mu(d) & \text{for } a' = a; \\ 0 & \text{for } a' > a. \end{cases}$$

Clearly, 0 is an absorbing state. We define the random variable $T = \min\{t \mid R_t = 0\}$. Let us write $\mathbf{E}_\mu(T)$ for the expectation of T if R_0 is uniformly distributed in $A - \{0\} = [1, n]$. We study $\mathbf{E}_\mu(T)$ in dependence on μ. In particular, we wish to identify distributions μ that make $\mathbf{E}_\mu(T)$ as small as possible (up to constant factors, where n is growing).

Observation 1.2. If $\mu(1) = 0$ then $\mathbf{E}_\mu(T) = \infty$.

This is because with probability $\frac{1}{n}$ position 1 is chosen as the starting point, and from state 1, the process will never reach 0 if $\mu(1) = 0$. As a consequence, for the whole paper we assume that all distributions μ that are considered satisfy

$$\mu(1) > 0. \tag{1.2}$$

Next we note that it is not hard to derive a "closed expression" for $\mathbf{E}_\mu(T)$. Fix μ. For $a \in A$, let $F(a) = \mu([1, a]) = \sum_{1 \leq d \leq a} \mu(d)$. We note recursion formulas for the expected travel time $T_a = \mathbf{E}_\mu(\, T \mid R_0 = a\,)$ when starting from position $a \in A$. It is not hard to

obtain (details are omitted due to space constraints)

$$\mathbf{E}_\mu(T) = \frac{1}{n} \cdot \sum_{1 \le a_1 < \cdots < a_\ell \le n} \frac{\mu(a_2 - a_1) \cdots \mu(a_\ell - a_{\ell-1})}{F(a_1) \cdots F(a_\ell)}, \tag{1.3}$$

where the sum ranges over all $2^n - 1$ nonempty subintervals $[a_1, a_\ell]$ of $[1, n]$. By definition of $F(a)$, we see that $\mathbf{E}_\mu(T)$ is a rational function of $(\mu(1), \ldots, \mu(n))$. By compactness, there is some μ that minimizes $\mathbf{E}_\mu(T)$. Unfortunately, there does not seem to be an obvious way to use (1.3) to gain information about the way $\mathbf{E}_\mu(T)$ depends on μ or what a distribution μ that minimizes $\mathbf{E}_\mu(T)$ looks like.

2. Upper bound

In this section, we establish upper bounds on $\mathbf{E}_\mu(T)$. We split the state space A and the set X of possible distances into "orders of magnitude", arbitrarily choosing 2 as the base.[1] Let $L = \lfloor \log n \rfloor$, and define $I_i = [2^i, 2^{i+1})$, for $0 \le i < L$, and $I_L = [2^L, n]$. Define

$$p_i = \sum_{d \in I_i} \mu(d), \text{ for } 0 \le i \le L.$$

Clearly, then, $p_0 + p_1 + \cdots + p_L = 1$. To simplify notation, we do not exclude terms that mean p_i for $i < 0$ or $i > L$. Such terms are always meant to have value 0. Consider the process $R = (R_0, R_1, \ldots)$. Assume $t \ge 1$ and $i \ge 1$. If $R_{t-1} \ge 2^i$ then all numbers $d \in I_{i-1}$ will be accepted as steps and lead to a progress of at least 2^{i-1}. Hence

$$\mathbf{Pr}(R_t \le R_{t-1} - 2^{i-1} \mid R_{t-1} \ge 2^i) \ge p_{i-1}.$$

Further, if $R_{t-1} \in I_i$, we need to choose step sizes from I_{i-1} at most twice to get below 2^i. Since the expected waiting time for the random distances to hit I_{i-1} twice is $2/p_{i-1}$, the expected time process R remains in I_i is not larger than $2/p_{i-1}$.

Adding up over $1 \le i \le L$, the expected time process R spends in the interval $[2, a]$, where $a \in I_j$ is the starting position, is not larger than

$$\frac{2}{p_{j-1}} + \frac{2}{p_{j-2}} + \ldots + \frac{2}{p_1} + \frac{2}{p_0}\ .$$

After the process has left $I_1 = [2, 3]$, it has reached position 0 or position 1, and the expected time before we hit 0 is not larger than $1/p_0 = 1/\mu(1)$. Thus, the expected number T_a of steps to get from $a \in I_j$ to 0 satisfies $T_a \le \frac{2}{p_{j-1}} + \frac{2}{p_{j-2}} + \ldots + \frac{2}{p_1} + \frac{3}{p_0}$. This implies the bound

$$\mathbf{E}_\mu(T) \le \frac{2}{p_{L-1}} + \frac{2}{p_{L-2}} + \ldots + \frac{2}{p_1} + \frac{3}{p_0}\ ,$$

for arbitrary μ. If we arrange that

$$p_0 = \cdots = p_{L-1} = \frac{1}{L}, \tag{2.1}$$

we will have $T_a \le (2j+1)L \le (2(\log a) + 1)(\log n) = O((\log a)(\log n)) = O((\log n)^2)$. Clearly, then, $\mathbf{E}_\mu(T) = O((\log n)^2)$ as well. The simplest distribution μ with (2.1) is the one that distributes the weight evenly on the powers of 2 below 2^L:

$$\mu_{\mathrm{pow2}}(d) = \begin{cases} 1/L, & \text{if } d = 2^i,\ 0 \le i < L, \\ 0, & \text{otherwise.} \end{cases}$$

[1] log means "logarithm to the base 2" throughout.

Thus, $\mathbf{E}_{\mu_{\text{pow2}}}(T) = O((\log n)^2)$. The "harmonic distribution" defined by (1.1) satisfies $p_i \approx (\ln(2^{i+1}) - \ln(2^i))/H_n \approx \ln 2/\ln(n) = 1/\log_2 n$, and we also get $T_a = O((\log a)(\log n))$ and $\mathbf{E}_{\mu_{\text{har}}}(T) = O((\log n)^2)$. More generally, all distributions μ with $p_0, \ldots, p_{L-1} \geq \alpha/L$, where $\alpha > 0$ is constant, satisfy $\mathbf{E}_\mu(T) = O((\log n)^2)$.

3. Lower bound

We show, as the main result of this paper, that the upper bound of Section 2 is optimal up to a constant factor.

Theorem 3.1. $\mathbf{E}_\mu(T) = \Omega((\log n)^2)$ *for all distributions* μ.

This theorem is proved in the remainder of this section. The distribution μ is fixed from here on; we suppress μ in the notation. Recall that we may assume that $\mu(1) > 0$. We continue to use the intervals $I_0, I_1, I_2, \ldots, I_L$ that partition $[1, n]$, as well as the probabilities p_i, $0 \leq i \leq L$.

3.1. Intuition

The basic idea for the lower bound is the following. For the majority of the starting positions, the process has to traverse all intervals $I_{L-2}, I_{L-3}, \ldots, I_1, I_0$. Consider an interval I_i. If the process reaches interval I_{i+1}, then afterwards steps of size 2^{i+2} and larger are rejected, and so do not help at all for crossing I_i. Steps of size from I_{i+1}, I_i, I_{i-1}, I_{i-2} may be of significant help. Smaller step sizes will not help much. So, very roughly, the expected time to traverse interval I_i completely when starting in I_{i+1} will be bounded from below by

$$\frac{1}{p_{i+1} + p_i + p_{i-1} + p_{i-2}},$$

since $1/(p_{i+1} + p_i + p_{i-1} + p_{i-2})$ is the waiting time for the first step with a "significant" size to appear. If it were the case that there is a constant $\beta > 0$ with the property that for each $0 \leq i < L - 1$ the probability that interval I_{i+1} is visited is at least β then it would not be hard to show that the expected travel time is bounded below by

$$\sum_{1 \leq j < L/2} \frac{\beta}{p_{2j+1} + p_{2j} + p_{2j-1} + p_{2j-2}}. \tag{3.1}$$

(We picked out only the even $i = 2j$ to avoid double counting.) Now the sum of the denominators in the sum in (3.1) is at most 2, and the sum is minimal when all denominators are equal, so the sum is bounded below by $\beta \cdot (L/2) \cdot (L/2)/2 = \beta \cdot L^2/8$, hence the expected travel time would be $\Omega(L^2) = \Omega((\log n)^2)$.

It turns out that it is not straightforward to turn this informal argument into a rigorous proof. First, there are (somewhat strange) distributions μ for which it is not the case that each interval is visited with constant probability. (For example, let $\mu(d) = B^{d-1} \cdot (B-1)/(B^n - 1)$, for a large base B like $B = n^3$. Then the "correct" jump directly to 0 has an overwhelming probability to be chosen first.[2]) Even for reasonable distributions μ, it may happen that some intervals or even blocks of intervals are jumped over with high probability. This means that the analysis of the cost of traversing I_i has to take into account that this traversal might happen in one big jump starting from an interval I_j with j much

[2]The authors thank Uri Feige for pointing this out.

larger than i. Second, in a formal argument, the contribution of the steps of size smaller than 2^{i-2} must be taken into account.

In the remainder of this section, we give a rigorous proof of the lower bound. For this, some machinery has to be developed. The crucial components are a reformulation of process R as another process, which as long as possible defers decisions about what the (randomly chosen) starting position is, and a potential function to measure how much progress the process has made in direction to its goal, namely reaching position 0.

3.2. Reformulation of the process

We change our point of view on the process R (with initial distribution uniform in $[1, n]$). The idea is that we do not have to fix the starting position right at the beginning, but rather make partial decisions on what the starting position is as the process advances. The information we hold on for step t is a random variable S_t, with the following interpretation: if $S_t > 0$ then R_t is uniformly distributed in $[1, S_t]$; if $S_t = 0$ then $R_t = 0$.

What properties should the random process $S = (S_0, S_1, \ldots)$ on $[0, n]$ have to be a proper model of the Markov chain R from Section 1.2? We first give an intuitive description of process S, and later formally define the corresponding Markov chain. Clearly, $S_0 = n$: the starting position is uniformly distributed in $[1, n]$. Given $s = S_{t-1} \in [0, n]$, we choose a step length d from X, according to distribution μ. Then there are two cases.

Case 1: $d > s$. — If $s \geq 1$, this step cannot be used for any position in $[1, s]$, thus we reject it and let $S_t = s$. If $s = 0$, no further move is possible at all, and we also reject.

Case 2: $d \leq s$. — Then $s \geq 1$, and the token is at some position in $[1, s]$. What happens now depends on the position of the token relative to d, for which we only have a probability distribution. We distinguish three subcases:

(i) The position of the token is larger than d. — This happens with probability $(s-d)/s$. In this case we "accept" the step, and now know that the token is in $[1, s-d]$, uniformly distributed; thus, we let $S_t = s - d$.

(ii) The position of the token equals d. — This happens with probability $1/s$. In this case we "finish" the process, and let $S_t = 0$.

(iii) The position of the token is smaller than d. — This happens with probability $\frac{d-1}{s}$. In this case we "reject" the step, and now know that the token is in $[1, d-1]$, uniformly distributed; thus, we let $S_t = d - 1$.

Clearly, once state 0 is reached, all further steps are rejected via Case 1.

We formalize this idea by defining a new Markov chain $S = (S_0, S_1, \ldots)$, as follows. The state space is $A = [0, n]$. For a state s', we collect the total probability that we get from s to s'. If $s' > s$, this probability is 0; if $s' = s$, this probability is $\sum_{s<d\leq n} \mu(d) = 1 - F(s)$; if $s' = 0$, this probability is $\sum_{1\leq d\leq s} \mu(d)/s = F(s)/s$; if $1 \leq s' < s$, this probability is $(\mu(s'+1) + \mu(s-s')) \cdot s'/s$, since d could be $s'+1$ or $s-s'$. Thus, we have the following transition probabilities:

$$p_{s,s'} = \begin{cases} F(s)/s & \text{if } s > s' = 0; \\ (\mu(s'+1) + \mu(s-s')) \cdot s'/s & \text{if } s > s' \geq 1; \\ 1 - F(s) & \text{if } s = s'. \end{cases}$$

Again, several initial distributions are possible for process S. The version with initial distribution with $\mathbf{Pr}(S_0 = n) = 1$ is meant to describe process R. Define the stopping time

$$T_S = \min\{t \mid S_t = 0\}.$$

We note that it is sufficient to analyze process S (with the standard initial distribution).

Lemma 3.2. $\mathbf{E}(T) = \mathbf{E}(T_S)$.

Proof. For $0 \leq s \leq n$, consider the version $R^{(s)}$ of process R induced by choosing the uniform distribution on $[1, s]$ (for $s \geq 1$) resp. $\{0\}$ (for $s = 0$) as the initial distribution. We let

$$A^{(s)} = \mathbf{E}(\min\{t \mid R_t^{(s)} = 0\}).$$

Clearly, $A^{(n)} = \mathbf{E}(T)$ and $A^{(0)} = 0$. We derive a recurrence for $(A^{(0)}, \ldots, A^{(n)})$. Let $s \geq 1$, and assume the starting point R_0 is chosen uniformly at random from $[1, s]$. We carry out the first step of $R^{(s)}$, which starts with choosing d. The following situations may arise.

(i) $d > s$. — This happens with probability $1 - F(s) < 1$. This distance will be rejected for all starting points in $[1, s]$, so the expected remaining travel time is $A^{(s)}$ again.

(ii) $1 \leq d \leq s$. For each d, the probability for this to happen is $\mu(d)$. For the starting point R_0 there are three possibilities:
- $R_0 \in [1, d-1]$ (only possible if $d > 1$). — This happens with probability $\frac{d-1}{s}$. The remaining expected travel time is $A^{(d-1)}$.
- $R_0 = d$. — This happens with probability $\frac{1}{s}$. The remaining travel time is 0.
- $R_0 \in [d+1, s]$ (only possible if $d < s$). — This happens with probability $\frac{s-d}{s}$. The remaining expected travel time in this case is $A^{(s-d)}$.

We obtain:

$$A^{(s)} = 1 + (1 - F(s))A^{(s)} + \sum_{1 \leq d \leq s} \mu(d) \left(\frac{d-1}{s} \cdot A^{(d-1)} + \frac{s-d}{s} \cdot A^{(s-d)} \right).$$

We rename $d - 1$ into s' in the first sum and $s - d$ into s' in the second sum and rearrange to obtain

$$A^{(s)} = \frac{1}{F(s)} \cdot \left(1 + \sum_{1 \leq s' < s} (\mu(s'+1) + \mu(s-s')) \cdot (s'/s) \cdot A^{(s')} \right). \tag{3.2}$$

Next, we consider process S. For $0 \leq s \leq n$, let $S^{(s)}$ be the process obtained from S by choosing s as the starting point. Clearly, $S^{(0)}$ always sits in 0, and $S^{(n)}$ is just S. Let

$$B^{(s)} = \mathbf{E}(\min\{t \mid S_t^{(s)} = 0\}),$$

the expected number of steps process S needs to reach 0 when starting in s. Then $B^{(0)} = 0$ and $B^{(n)} = \mathbf{E}(T_S)$. We derive a recurrence for $(B^{(0)}, \ldots, B^{(n)})$. Let $s \geq 1$. Carry out the first step of $S^{(s)}$, which leads to state s'. The following situations may arise.

(i) $s = s' \geq 1$. — This occurs with probability $1 - F(s)$, and the expected remaining travel time is $B^{(s)}$ again.

(ii) $s' = 0$. — In this case the expected remaining travel time is $B^{(0)} = 0$.

(iii) $s > s' \geq 1$. — This occurs with probability $(\mu(s'+1) + \mu(s-s')) \cdot s'/s$. The expected remaining travel time is $B^{(s')}$.

Summing up, we obtain

$$B^{(s)} = 1 + (1 - F(s))B^{(s)} + \sum_{1 \leq s' < s} (\mu(s'+1) + \mu(s-s')) \cdot (s'/s) \cdot B^{(s')}.$$

Solving for $B^{(s)}$ yields:

$$B^{(s)} = \frac{1}{F(s)} \cdot \left(1 + \sum_{1 \le s' < s} (\mu(s'+1) + \mu(s-s')) \cdot (s'/s) \cdot B^{(s')} \right). \tag{3.3}$$

Since $A^{(0)} = 0 = B^{(0)}$ and the recurrences (3.2) and (3.3) are identical, we have $\mathbf{E}(T) = A^{(n)} = B^{(n)} = \mathbf{E}(T_S)$, as claimed. ∎

3.3. Potential function: Definition and application

We introduce a potential function Φ on the state space $A = [0, n]$ to bound the progress of process S. Our main lemma states that for any $s \in A$, for a random transition from $S_i = s$ to S_{i+1} the expected loss in potential is at most constant (i.e., $\mathbf{E}(\Phi(S_{i+1}) - \Phi(S_i) \mid S_i = s) = O(1)$). This implies that $\mathbf{E}(T_S) = \Omega(\Phi(S_0))$. Since the potential function will satisify $\Phi(S_0) = \Omega(\log^2 n)$, the lower bound follows.

We start by trying to give intuition for the definition. A rough approximation to the potential function we use would be the following: For interval I_i there is a term

$$\psi_i = \frac{1}{\sum_{0 \le j \le L} p_j \cdot c^{|j-i|}}, \tag{3.4}$$

for some constant c with $\frac{1}{2} < c < 1$, e. g., $c = 1/\sqrt{2}$. For later use we note that

$$\sum_{1 \le i < L} \psi_i^{-1} = \sum_{1 \le i < L} \sum_{0 \le j \le L} p_j \cdot c^{|j-i|} = \sum_{0 \le j \le L} p_j \sum_{1 \le i < L} c^{|j-i|} = O(1), \tag{3.5}$$

since $\sum_{0 \le j \le L} p_j = 1$ and $\sum_{k \ge 0} c^k = \frac{1}{1-c}$. The term ψ_i tries to give a rough lower bound for the expected number of steps needed to cross I_i in the following sense: The summands $p_j \cdot c^{|j-i|}$ reflect the fact that step sizes that are close to I_i will be very helpful for crossing I_i, and step sizes far away from I_i might help a little in crossing I_i, but they do so only to a small extent ($j \ll i$) or with small probability ($j \gg i$). The idea is then to arrange that a state $s \in I_k$ has potential about

$$\Psi_k = \sum_{i \le k} \psi_i. \tag{3.6}$$

It turns out that analyzing process S on the basis of a potential function that refers to the intervals I_i is possible but leads to messy calculations and numerous cases. The calculations become cleaner if one avoids the use of the intervals in the definition and in applying the potential function. The following definition derives from (3.4) and (3.6) by splitting up the summands ψ_i into contributions from all positions $a \in I_i$ and smoothing out the factors $c^{|j-i|} = 2^{|j-i|/2}$, for $a \in I_i$ and $d \in I_j$, into $2^{-|\log a - \log d|/2}$, which is $\sqrt{a/d}$ for $a \le d$ and $\sqrt{d/a}$ for $d \le a$. This leads to the following[3]. Assumption (1.2) guarantees that in the formulas to follow all denominators are nonzero.

Definition 3.3. For $1 \le a \le n$ let

$$\sigma_a = \sum_{1 \le d \le n} \mu(d) \cdot 2^{-|\log a - \log d|/2} = \sum_{1 \le d \le a} \mu(d) \sqrt{\frac{d}{a}} + \sum_{a < d \le n} \mu(d) \sqrt{\frac{a}{d}}$$

[3]Whenever in the following we use letters a, b, d, the range $[1, n]$ is implicitly understood.

and $\varphi_a = 1/(a\sigma_a)$. For $0 \le s \le n$ define $\Phi(s) = \sum_{1\le a\le s} \varphi_a$. The random variable Φ_t, $t = 0, 1, 2, \ldots$, is defined as $\Phi_t = \Phi(S_t)$.

We note some easy observations and one fundamental fact about Φ_t, $t \ge 0$.

Lemma 3.4.

(a) Φ_t, $t \ge 0$, *is nonincreasing for* t *increasing.*
(b) $\Phi_t = 0 \Leftrightarrow S_t = 0$.
(c) $\Phi_0 = \Omega((\log n)^2)$ (Φ_0 *is a number that depends on* n *and* μ).

Proof. (a) is clear since S_t, $t \ge 0$, is nonincreasing and the terms φ_a are positive. — (b) is obvious since $\Phi_t = 0$ if and only if $\Phi(S_t)$ is the empty sum, which is the case if and only if $S_t = 0$. — We prove (c). In this proof we use the intervals I_i and the probabilities p_i, $0 \le i \le L$, from Section 2. We use the notation $i(a) = \lfloor \log a \rfloor = \max\{i \mid 2^i \le a\}$. We start with finding an upper bound for σ_a by grouping the summands in σ_a according to the intervals. Let $c = 1/\sqrt{2}$.

$$\begin{aligned}
\sigma_a &= \sum_{1\le d\le n} \mu(d) \cdot 2^{-|\log a - \log d|/2} \\
&\le \sum_{j\le i(a)} \sum_{d\in I_j} \mu(d) \cdot 2^{(j+1-i(a))/2} + \sum_{j>i(a)} \sum_{d\in I_j} \mu(d) \cdot 2^{(i(a)+1-j)/2} \\
&= \sum_{j\le i(a)} p_j \cdot 2^{(j+1-i(a))/2} + \sum_{j>i(a)} p_j \cdot 2^{(i(a)+1-j)/2} = 2c \cdot \left(\sum_{0\le j\le L} p_j \cdot c^{|j-i(a)|} \right).
\end{aligned}$$

Hence

$$\sum_{a\in I_i} \varphi_a = \sum_{a\in I_i} \frac{1}{a\sigma_a} \ge \frac{2^i}{2c \cdot 2^{i+1} \cdot \left(\sum_{0\le j\le L} p_j \cdot c^{|j-i|} \right)} = \frac{\psi_i}{4c},$$

with ψ_i from (3.4). Thus,

$$\Phi_0 \ge \sum_{0\le i<L} \frac{\psi_i}{4c}. \tag{3.7}$$

Let $u_i = 4c/\psi_i$ be the reciprocal of the summand for i in (3.7), $0 \le i < L$. From (3.5) we read off that $\sum_{0\le i<L} u_i \le k$, for some constant k. Now $\sum_{0\le i<L} \frac{1}{u_i}$ with $\sum_{0\le i<L} u_i \le k$ is minimal if all u_i are equal to k/L. Together with (3.7) this entails $\Phi_0 \ge L \cdot (L/k) = L^2/k = \Omega((\log n)^2)$, which proves part (c) of Lemma 3.4. ∎

The crucial step in the lower bound proof is to show that the progress made by process S in one step, measured in terms of the potential, is bounded:

Lemma 3.5 (Main Lemma). *There is a constant* C *such that for* $0 \le s \le n$, *we have* $\mathbf{E}(\Phi_{t-1} - \Phi_t \mid S_{t-1} = s) \le C$.

The proof of Lemma 3.5 is the core of the analysis. It will be given in Section 3.4. To prove Theorem 3.1, we need the following lemma, which is stated and proved (as Lemma 12) in [2]. (It is a one-sided variant of Wald's identity.)

Lemma 3.6. *Let* $X_1, X_2, \ldots$ *denote random variables with bounded range, let* $g > 0$ *and let* $T = \min\{t \mid X_1 + \cdots + X_t \ge g\}$. *If* $\mathbf{E}(T) < \infty$ *and* $\mathbf{E}(X_t \mid T \ge t) \le C$ *for all* $t \in \mathbb{N}$, *then* $\mathbf{E}(T) \ge g/C$.

Proof of 3.1: Since $S_t = 0$ if and only if $\Phi_t = 0$ (Lemma 3.4(b)), the stopping time $T_\Phi = \min\{t \mid \Phi_t = 0\}$ of the potential reaching 0 satisfies $T_\Phi = T_S$. Thus, to prove Theorem 3.1, it is sufficient to show that $\mathbf{E}(T_\Phi) = \Omega((\log n)^2)$. For this, we let $X_t = \Phi_{t-1} - \Phi_t$, the progress made in step t in terms of the potential. By Lemma 3.5, $\mathbf{E}(X_t \mid S_{t-1} = s) \le C$, for all $s \ge 1$, and hence

$$\mathbf{E}(X_t \mid T \ge t) = \mathbf{E}(X_t \mid \Phi(S_{t-1}) > 0) \le C\,.$$

Observe that $X_1 + \cdots + X_t = \Phi_0 - \Phi_t$ and hence $T_\Phi = \min\{t \mid X_1 + \cdots + X_t \ge \Phi_0\}$. Applying Lemma 3.6, and combining with Lemma 3.4, we get that $\mathbf{E}(T_\Phi) \ge \Phi_0/C = \Omega((\log n)^2)$, which proves Theorem 3.1. ∎

The only missing part to fill in is the proof of Lemma 3.5.

3.4. Proof of the Main Lemma (Lemma 3.5)

Fix $s \in [1, n]$, and assume $S_{t-1} = s$. Our aim is to show that the "expected potential loss" is constant, i. e., that

$$\mathbf{E}(\Phi_t - \Phi_{t-1} \mid S_{t-1} = s) = O(1).$$

Clearly, $\mathbf{E}(\Phi_t - \Phi_{t-1} \mid S_{t-1} = s) = \sum_{0 \le x \le s} \Delta(s, x)$, where

$$\Delta(s, x) = \big(\Phi(s) - \Phi(x)\big) \cdot \mathbf{Pr}(S_t = x \mid S_{t-1} = s). \tag{3.8}$$

We show that $\sum_{0 \le x \le s} \Delta(s, x)$ is bounded by a constant, by considering $\Delta(s, s)$, $\Delta(s, 0)$, and $\sum_{1 \le x < s} \Delta(s, x)$ separately.

For $x = s$, the potential difference $\Phi(s) - \Phi(x)$ is 0, and thus

$$\Delta(s, s) = 0. \tag{3.9}$$

Bounding $\Delta(s, 0)$: According to the definition of the process S, a step from $S_{t-1} = s$ to $S_t = 0$ has probability $F(s)/s$. Since $\Phi(0) = 0$, the potential difference is $\Phi(s)$. Thus, we obtain

$$\Delta(s,0) = \frac{1}{s} \cdot \left(\sum_{d \le s} \mu(d)\right) \cdot \left(\sum_{a \le s} \varphi_a\right) = \frac{1}{s} \cdot \sum_{a \le s} \frac{\displaystyle\sum_{d \le s} \mu(d)}{\displaystyle\sum_{b \le a} \mu(b)\sqrt{ab} + \sum_{a < b \le n} \mu(b) a^{3/2}/\sqrt{b}}$$

$$\le \frac{1}{s} \cdot \sum_{a \le s} \delta(a), \quad \text{where } \delta(a) \;=\; \frac{\displaystyle\sum_{b \le s} \mu(b)}{\displaystyle\sum_{b \le a} \mu(b)\sqrt{ab} + \sum_{a < b \le s} \mu(b) a^{3/2}/\sqrt{b}}\,.$$

We bound $\delta(a)$. For $b \le a$ and $\mu(b) \ne 0$, the quotient of the summands in the numerator and denominator of $\delta(a)$ that correspond to b is $1/\sqrt{ab} \le \sqrt{a}/a \le \sqrt{s}/a$. For $a < b$ and $\mu(b) \ne 0$, the quotient is $\sqrt{b}/a^{3/2} \le \sqrt{s}/a$. Thus, $\delta(a) \le \frac{\sqrt{s}}{a}$. This implies (recall that $H_s = \sum_{a \le d \le s} \frac{1}{a}$):

$$\Delta(s,0) \le \frac{1}{s} \cdot \sum_{a \le s} \sqrt{s}/a \le \frac{H_s}{\sqrt{s}} \le \frac{\ln(s) + 1}{\sqrt{s}} < 2. \tag{3.10}$$

Bounding $\sum_{1\le x<s}\boldsymbol{\Delta(s,x)}$: Assume $1 \le x < s$. According to the definition of the process S,

$$\mathbf{Pr}(S_{t-1} = x \mid S_t = s) = \frac{x}{s}\cdot\big(\mu(x+1)+\mu(s-x)\big).$$

The potential difference is $\Phi(s)-\Phi(x) = \sum_{x<a\le s}\varphi_a$. Thus we have

$$\sum_{1\le x<s}\Delta(s,x) = \sum_{1\le x<s}\sum_{x<a\le s}\varphi_a\cdot\frac{x}{s}\cdot\big(\mu(x+1)+\mu(s-x)\big) = \frac{1}{s}\cdot\sum_{1<a\le s}(\lambda_a+\gamma_a), \qquad (3.11)$$

where $\lambda_a = \varphi_a\cdot\sum_{1\le x<a}\mu(x+1)\,x$ and $\gamma_a = \varphi_a\cdot\sum_{1\le x<a}\mu(s-x)\,x$. We bound λ_a and γ_a separately. Observe first that

$$\begin{aligned}\lambda_a &= \varphi_a\cdot\sum_{2\le x\le a}\mu(x)(x-1)\\ &\le \frac{\displaystyle\sum_{1\le x\le a}\mu(x)(x-1)}{\displaystyle\sum_{1\le b\le a}\mu(b)\cdot\sqrt{ab}+\sum_{a<b\le n}\mu(b)a^{3/2}/\sqrt{b}} \le \frac{\displaystyle\sum_{1\le b\le a}\mu(b)(b-1)}{\displaystyle\sum_{1\le b\le a}\mu(b)\sqrt{ab}\,.}\end{aligned} \qquad (3.12)$$

(We used the definition of φ_a, and omitted some summands in the denominator.) Recall that $\mu(1) > 0$, so the denominator is not zero. For each $b \le a$ we clearly have $\mu(b)(b-1) \le \mu(b)\sqrt{ab}$, thus the sum in the numerator in (3.12) is smaller than the sum in the denominator, and we get $\lambda_a < 1$.

Next, we bound γ_a for $a \le s$:

$$\begin{aligned}\gamma_a = \varphi_a\cdot\sum_{1\le x<a}\mu(s-x)\,x &= \varphi_a\cdot\sum_{s-a<x<s}\mu(x)\,(s-x)\\ &= \frac{\displaystyle\sum_{s-a<x\le a}\mu(x)(s-x)+\sum_{\max\{a,s-a\}<x<s}\mu(x)(s-x)}{\displaystyle\sum_{1\le b\le a}\mu(b)\sqrt{ab}+\sum_{a<b\le n}\mu(b)a^{3/2}/\sqrt{b}}\,.\end{aligned}$$

The denominator is not zero because $\mu(1) > 0$. Hence, if $\mu(x) = 0$ for all $s-a < x < s$, then $\gamma_a = 0$. Otherwise, by omitting some of the summands in the denominator we obtain

$$\gamma_a \le \frac{\displaystyle\sum_{s-a<b\le a}\mu(b)\,(s-b)+\sum_{\max\{a,s-a\}<b<s}\mu(b)\,(s-b)}{\displaystyle\sum_{s-a<b\le a}\mu(b)\sqrt{ab}+\sum_{\max\{a,s-a\}<b<s}\mu(b)a^{3/2}/\sqrt{b}}$$

(If $a \le s/2$, the first sum in both numerator and denominator is empty.) Now consider the quotient of the summands for each b with $\mu(b) > 0$. For $s-a < b \le a$, this quotient is

$$\frac{\mu(b)\,(s-b)}{\mu(b)\sqrt{ab}} \le \frac{a-1}{\sqrt{a\cdot(s-a+1)}} < \sqrt{\frac{a}{s-a+1}} \le \sqrt{\frac{s}{s-a+1}}.$$

For $\max\{a, s-a\} < b < s$, the quotient of the corresponding summands is

$$\frac{\mu(b)(s-b)}{\mu(b)a^{3/2}/\sqrt{b}} \le \frac{\min\{a,s-a\}\cdot\sqrt{b}}{a^{3/2}} \le \frac{a\cdot\sqrt{s}}{a^{3/2}} = \sqrt{\frac{s}{a}}.$$

Hence, $\gamma_a \le \sqrt{s/(s-a+1)} + \sqrt{s/a}$. Plugging this bound on γ_a and the bound $\lambda_a < 1$ into (3.11), and using that

$$\sum_{1\le a\le s} \frac{1}{\sqrt{a}} = 1 + \sum_{2\le a\le s} \frac{1}{\sqrt{a}} < 1 + \int_1^s \frac{dx}{\sqrt{x}} = 1 + [2\sqrt{x}]_1^s = 1 + 2\sqrt{s} - 2 < 2\sqrt{s},$$

we obtain

$$\sum_{1\le x<s} \Delta(s,x) < \frac{1}{s} \cdot \sum_{1<a\le s} \left(1 + \sqrt{\frac{s}{a}} + \sqrt{\frac{s}{s-a+1}}\right)$$

$$< 1 + \frac{1}{\sqrt{s}} \left(\sum_{1<a\le s} \sqrt{\frac{1}{a}} + \sum_{1\le a<s} \sqrt{\frac{1}{a}} \right) < 1 + \frac{2}{\sqrt{s}} \sum_{1\le a\le s} \frac{1}{\sqrt{a}} < 1 + \frac{2}{\sqrt{s}} \cdot 2\sqrt{s} = 5. \quad (3.13)$$

Summing up the bounds from (3.9), (3.10), and (3.13), we obtain

$$\mathbf{E}(\Phi_t - \Phi_{t-1} \mid S_{t-1} = s) \le \Delta(s,0) + \sum_{1\le x<s} \Delta(s,x) + \Delta(s,s) < 2 + 5 + 0 = 7.$$

Thus Lemma 3.5 is proved. ∎

4. Open problems

1. We conjecture that the method can be adapted to the continuous case to prove a lower bound of $\Omega((\log(1/\varepsilon)^2)$ for approximating the minimum of some unimodal function f by a scale-invariant search strategy (see Section 1.1).

2. It is an open problem whether our method can be used to prove a lower bound of $\Omega((\log n)^2)$ for finding the minimum of an arbitrary unimodal function $f\colon \{0,\dots,n\} \to \mathbb{R}$ by a scale invariant search strategy.

Acknowledgement

The authors thank two anonymous referees for their careful reading of the manuscript and for providing several helpful comments.

References

[1] Droste, S., Jansen, T., and Wegener, I., Upper and lower bounds for randomized search heuristics in black-box optimization, *Theory Comput. Syst.* **39**(4) 525–544 (2006).

[2] Jägersküpper, J., Algorithmic analysis of a basic evolutionary algorithm for continuous optimization, *Theor. Comput. Sci.* **379**(3) 329–347 (2007).

[3] Kiefer, J., Sequential minimal search for a maximum, *Proc. Amer. Math. Soc.* **4** 502–506 (1953).

[4] Rowe, J. E., and Hidović, D., An evolution strategy using a continuous version of the Gray-code neighbourhood distribution, in: K. Deb *et al.*, *Eds.*, Proc. GECCO 2004, Part 1, LNCS Vol. 3102, Springer-Verlag, pp. 725–736.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 253-264
www.stacs-conf.org

DISCRETE JORDAN CURVE THEOREM: A PROOF FORMALIZED IN COQ WITH HYPERMAPS

JEAN-FRANÇOIS DUFOURD [1]

[1] Université Louis-Pasteur de Strasbourg, UFR de Mathématique et d'Informatique,
Labo. des Sciences de l'Image, de l'Informatique et de la Télédétection (UMR CNRS-ULP 7005),
Pôle API, Boulevard Sébastien Brant, 67400 Illkirch, France
E-mail address: dufourd@dpt-info.u-strasbg.fr

Abstract. This paper presents a formalized proof of a discrete form of the Jordan Curve Theorem. It is based on a hypermap model of planar subdivisions, formal specifications and proofs assisted by the Coq system. Fundamental properties are proven by structural or noetherian induction: Genus Theorem, Euler's Formula, constructive planarity criteria. A notion of ring of faces is inductively defined and a Jordan Curve Theorem is stated and proven for any planar hypermap.

Introduction

This paper presents a formal statement and an assisted proof of a Jordan Curve Theorem (JCT) discrete version. In its common form, the theorem says that the complement of a continuous simple closed curve (a Jordan curve) C in an affine real plane is made of two connected components whose border is C, one being bounded and the other not. The discrete form of JCT we deal with states that in a finite subdivision of the plane, breaking a ring R of faces increases by 1 the connectivity of the subdivision. It is a weakened version of the original theorem where the question of bound is missing. However, it is widely used in computational geometry and discrete geometry for imaging, where connection is the essential information (14; 9). In fact, we only are in a combinatoric framework, where any embedding is excluded, and where bounding does not make sense.

In computational topology, subdivisions are best described by map models, the most general being *hypermaps* (15; 4). We propose a purely combinatorial proof of JCT based on this structure. The hypermap framework is entirely formalized and the proofs are developed interactively and verified by the Coq proof assistant (3). Using an original way to model, build and destruct hypermaps, the present work brings new simple constructive planarity and connectivity criteria. It proposes a new direct expression of JCT and a simple constructive proof with algorithmic extensions. It is also a large benchmark for the software

1998 ACM Subject Classification: I.3.5, E1, D.2.4.

Key words and phrases: Formal specifications - Computational topology - Computer-aided proofs - Coq - Planar subdivisions - Hypermaps - Jordan Curve Theorem.

Acknowledgements: This research is supported by the "white" project GALAPAGOS, French ANR, 2007.

specification framework we have been developing in the last fifteen years for map models used in geometric modeling and computer imagery (2; 7; 8).

The useful Coq features are reminded and the whole process is described, but the full details of the proofs are omitted. Section 1 summarizes related work. Section 2 recalls some mathematical materials. Section 3 proposes basic hypermap specifications. Section 4 proves constructive criteria of hypermap planarity and connectivity. Section 5 inductively specifies the rings and their properties. Section 6 proves the discrete JCT. Section 7 concludes.

1. Related work

The JCT is a result of classical plane topology, first stated by C. Jordan in 1887, but of which O. Veblen gives the first correct proof in 1905. In 1979, W.T. Tutte proposes operations and properties of combinatorial maps, *e.g.* planarity and Euler's Formula, defines rings and proves a discrete JCT (15). Our theorem statement is comparable, but our framework is modeled differently and all our proofs are formalized and computer-assisted.

In 2003, G. Bauer and T. Nipkow specify planar graphs and triangulations in Isabelle/Isar to carry out interactive proofs of Euler's Formula and of the Five Colour Theorem (1). However, they do not approach the JCT. In 2005, A. Kornilowicz designs for the MIZAR project a semi-automated classical proof of a continuous form of JCT in an Euclidean space (13). In 2005 also, on his way towards the proof of the Kepler conjecture in the Flyspeck projet, T. Hales proves the JCT for planar rectangular grids with the HOL Light system, following the Kuratowski characterization of planarity (12).

In 2005 always, G. Gonthier *et al.* prove the Four Colour Theorem using Coq. Plane subdivisions are described by hypermaps, and Euler's Formula is used as a global planarity criterion (10). A local criterion, called *hypermap Jordan property*, is proven equivalent. The main part of this work is the gigantic proof of the Four Colour Theorem with hypermaps and sophisticated proof techniques. The hypermap formalization is very different from ours and it seems that JCT is not explicitly proven there. Finally, since 1999, we carry out experiments with Coq for combinatorial map models of space subdivisions (5; 7; 8).

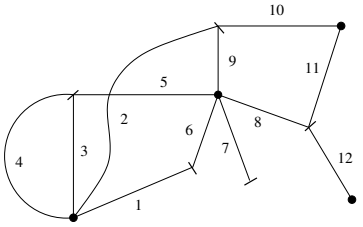
2. Mathematical Aspects

Definition 2.1 (Hypermap)**.** A *hypermap* is an algebraic structure $M = (D, \alpha_0, \alpha_1)$, where D is a finite set whose elements are called *darts*, and α_0, α_1 are permutations on D.

If $y = \alpha_k(x)$, y is the *k-successor* of x, x is the *k-predecessor* of y, and x and y are said to be *k-linked*.

In Fig. 1, as functions α_0 and α_1 on $D = \{1, \ldots, 15\}$ are permutations, $M = (D, \alpha_0, \alpha_1)$ is a hypermap. It is drawn on the plane by associating to each dart a curved arc oriented from a bullet to a small stroke: 0-linked (resp. 1-linked) darts share the same small stroke (resp. bullet). By convention, in the drawings of hypermaps on surfaces, k-successors turn counterclockwise around strokes and bullets. Let $M = (D, \alpha_0, \alpha_1)$ be a hypermap.

Definition 2.2. (Orbits and hypermap cells)
(1) Let $f_1, \ldots, f_n$ be n functions in D. The *orbit* of $x \in D$ for $f_1, \ldots, f_n$ is the subset of D denoted by $\langle f_1, \ldots, f_n \rangle(x)$, the elements of which are accessible from x by any composition of $f_1, \ldots, f_n$.



D	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
α0	6	9	5	3	4	1	7	12	10	2	8	11	14	13	15
α1	2	3	4	1	6	7	8	9	5	11	10	12	14	13	15

Figure 1: An example of hypermap.

(2) In M, $\langle\alpha_0\rangle(x)$ is the 0-orbit or *edge* of dart x, $\langle\alpha_1\rangle(x)$ its 1-orbit or *vertex*, $\langle\phi\rangle(x)$ its *face* for $\phi = \alpha_1^{-1} \circ \alpha_0^{-1}$, and $\langle\alpha_0, \alpha_1\rangle(x)$ its *(connected) component*.

In Fig. 1 the hypermap contains 7 edges (strokes), 6 vertices (bullets), 6 faces and 3 components. For instance, $\langle\alpha_0\rangle(3) = \{3, 5, 4\}$ is the edge of dart 3, $\langle\alpha_1\rangle(3) = \{3, 4, 1, 2\}$ its vertex. Faces are defined, through ϕ, for a dart traversal in counterclockwise order, when the hypermap is drawn on a surface. Then, every face which encloses a bounded (resp. unbounded) region on its left is called *internal* (resp. *external*). In Fig. 1, the (internal) face of 8 is $\langle\phi\rangle(8) = \{8, 10\}$ and the (external) face of 13 is $\langle\phi\rangle(13) = \{13\}$. Let d, e, v, f and c be the numbers of darts, edges, vertices, faces and components of M.

Definition 2.3. (Euler characteristic, genus, planarity)
(1) The *Euler characteristic* of M is $\chi = v + e + f - d$.
(2) The *genus* of M is $g = c - \chi/2$.
(3) When $g = 0$, M is said to be *planar*.

For instance, in Fig. 1, $\chi = 6 + 6 + 7 - 15 = 4$ and $g = 3 - \chi/2 = 1$. Consequently, the hypermap is non planar. These values satisfy the following results:

Theorem 2.4 (of the Genus)**.** *χ is an even integer and g is a natural number.*

Corollary 2.5 (Euler Formula)**.** *A non empty connected − i.e. with $c = 1$ − planar hypermap satisfies $v + e + f - d = 2$.*

When $D \neq \emptyset$, the *representation* of M on an *orientable closed* surface is a mapping of edges and vertices onto points, darts onto open oriented Jordan arcs, and faces onto open connected regions. It is an *embedding* when every component of M realizes a partition of the surface. Then, the genus of M is the minimum number of *holes* in an orientable closed surface where such an embedding is possible, thus drawing a subdivision, or a polyhedron, by hypermap component (11). For instance, all the components of the hypermap in Fig. 1 can be embedded on a torus (1 hole) but not on a sphere or on a plane (0 hole). When a (planar) hypermap component is embedded on a plane, the corresponding subdivision has exactly one unbounded (external) face. But a non planar hypermap can never be embedded on a plane: in a drawing on a plane, some of its faces are neither internal nor external, *e.g.* $\langle\phi\rangle(1) = \{1, 5, 2, 11, 12, 7, 6, 4, 9\}$ in Fig. 1. Conversely, any subdivision of an *orientable closed* surface can be modeled by a hypermap. In fact, the formal presentation which follows is *purely combinatorial*, *i.e* without any topological or geometrical consideration.

2.1. Rings of faces and Jordan Curve Theorem

To state the version of JCT we will prove, we need the concepts of *double-link*, *adjacent faces* and *ring of faces* in a hypermap $M = (D, \alpha_0, \alpha_1)$.

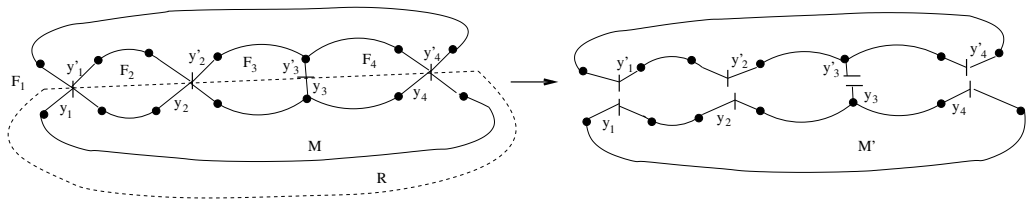


Figure 2: Break of M along a ring R of length $n = 4$ giving M'.

Definition 2.6. (Double-link and adjacent faces)
(1) A *double-link* is a pair of darts (y, y') where y and y' belong to the same edge.
(2) The faces F and F' of M are said to be *adjacent by the double-link* (y, y') if y is a dart of F and y' a dart of F'.

We choose a face adjacency *by an edge* rather than *by a vertex* as does W.T. Tutte (15). In fact, due to the homogeneity of dimensions 0 and 1 in a hypermap, both are equivalent.

Definition 2.7. (Ring of faces)
A *ring of faces* R of length n in M is a non empty sequence of double-links (y_i, y'_i), for $i = 1, \ldots, n$, with the following properties, where E_i and F_i are the edge and face of y_i:
(0) *Unicity:* E_i and E_j are distinct, for $i, j = 1, \ldots, n$ and $i \neq j$;
(1) *Continuity:* F_i and F_{i+1} are adjacent by the double-link (y_i, y'_i), for $i = 1, \ldots, n-1$;
(2) *Circularity*, or *closure:* F_n and F_1 are adjacent by the double-link (y_n, y'_n);
(3) *Simplicity:* F_i and F_j are distinct, for $i, j = 1, \ldots, n$ and $i \neq j$.

This notion simulates a Jordan curve represented in dotted lines in Fig. 2 on the left for $n = 4$. Then, we define the *break along a ring*, illustrated in Fig. 2 on the right.

Definition 2.8. (Break along a ring)
Let R be a ring of faces of length n in M. Let $M_i = (D, \alpha_{0,i}, \alpha_1)$, for $i = 0, \ldots, n$, be a hypermap sequence, where the $\alpha_{0,i}$ are recursively defined by:
(1) $i = 0$: $\alpha_{0,0} = \alpha_0$;
(2) $1 \leq i \leq n$: for each dart z of D: $\alpha_{0,i}(z) =$ if $\alpha_{0,i-1}(z) = y_i$ then y'_i else if $\alpha_{0,i-1}(z) = y'_i$ then y_i else $\alpha_{0,i-1}(z)$.
Then, $M_n = (D, \alpha_{0,n}, \alpha_1)$ is said to be obtained from M by a *break along* R.

Finally, the theorem we will prove in Coq mimics the behaviour of a cut along a simple Jordan curve of the plane (or of the sphere) into two components:

Theorem 2.9 (Discrete Jordan Curve Theorem)**.** *Let M be a planar hypermap with c components, R be a ring of faces in M, and M' be the break of M along R. The number c' of components of M' is such that $c' = c + 1$.*

3. Hypermap specifications

3.1. Preliminary specifications

In Coq, we first define an inductive type `dim` for the two dimensions at stake:

```
Inductive dim:Set:= zero: dim | one: dim.
```

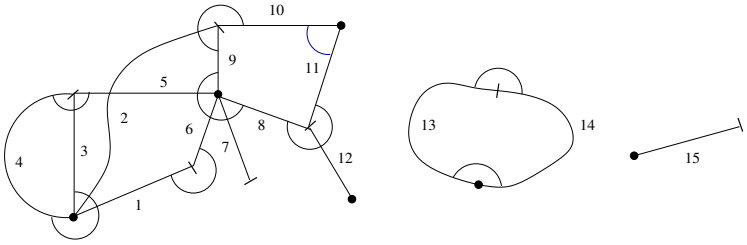


Figure 3: A hypermap with its incompletely linked orbits.

All objects being typed in Coq, `dim` has the type `Set` of all concrete types. Its *constructors* are the constants `zero` and `one`. In each inductive type, the generic equality predicate = is built-in but its decidability is not, because Coq's logic is intuitionistic. For `dim`, the latter can be established as the lemma:

```
Lemma eq_dim_dec: forall i j : dim, {i=j}+{~i=j}.
```

Once it is made, its proof is an object of the sum type `{i=j}+{~i=j}`, *i.e.* a function, named `eq_dim_dec`, that tests whenever its two arguments are equal. The lemma is interactively proven with some tactics, the reasoning being merely a structural induction on both `i` and `j`, here a simple case analysis. Indeed, from each inductive type definition, Coq generates an *induction principle*, usable either to prove propositions or to build total functions on the type. We identify the type `dart` and its equality decidability `eq_dart_dec` with the built-in `nat` and `eq_nat_dec`. Finally, to manage exceptions, a `nil` dart is a renaming of `0`:

```
Definition dart:= nat.
Definition eq_dart_dec:= eq_nat_dec.
Definition nil:= 0.
```

3.2. Free maps

The hypermaps are now approached by a general notion of *free map*, thanks to a free algebra of terms of inductive type `fmap` with 3 constructors, `V`, `I` and `L`, respectively for the *empty* (or *void*) map, the *insertion* of a dart, and the *linking* of two darts:

```
Inductive fmap:Set:=
   V : fmap | I : fmap->dart->fmap | L : fmap->dim->dart->dart->fmap.
```

For instance, the hypermap in Fig. 1 can be modeled by the free map represented in Fig.3 where the 0- and 1-links by `L` are represented by arcs of circle, and where the orbits remain open. Again, Coq generates an induction principle on free maps.

Next, *observers* of free maps can be defined. The predicate `exd` express that a dart exists in a hypermap. Its definition is recursive, which is indicated by `Fixpoint`, thanks to a pattern matching on `m` written `match m with...`. The attribute `{struct m}` allows Coq to verify that the recursive calls are performed on smaller `fmap` terms, thus ensuring termination. The result is `False` or `True`, basic constants of `Prop`, the built-in type of propositions. Note that terms are in prefix notation and that `_` is a place holder:

```
Fixpoint exd(m:fmap)(z:dart){struct m}:Prop:=
 match m with
   V => False | I m0 x => z=x \/ exd m0 z | L m0 _ _ _  => exd m0 z
 end.
```

The decidability `exd_dec` of `exd` directly derives, thanks to a proof by induction on `m`. Then, a version, denoted `A`, of operation α_k of Definition 2.1 completed with `nil` for convenience is written as follows, the inverse `A_1` being similar:

```
Fixpoint A(m:fmap)(k:dim)(z:dart){struct m}:dart:=
 match m with
   V => nil | I m0 x => A m0 k z | L m0 k0 x y =>
     if eq_dim_dec k k0 then if eq_dart_dec z x then y else A m0 k z
     else A m0 k z
 end.
```

Predicates `succ` and `pred` express that a dart has a `k`-successor and a `k`-predecessor (not `nil`), with the decidabilities `succ_dec` and `pred_dec`. In hypermap `m` of Fig. 3, `A m zero 4 = 3, A m zero 5 = nil, succ m zero 4 = True, succ m zero 5 = False, A_1 m one 2 = 1`. In fact, when a `k`-orbit remains open, which will be required in the following, we can obtain its `top` and `bottom` from one of its dart `z`. Then, we can do as if the `k`-orbit were closed, thanks to the operations `cA` and `cA_1` which *close* `A` and `A_1`, in a way similar to operation K of W.T. Tutte (15). For instance, in Fig. 3, `top m one 1 = 3, bottom m one 1 = 4, cA m one 3 = 4, cA_1 m one 4 = 3`.

Finally, *destructors* are also recursively defined. First, `D:fmap->dart->fmap` deletes the latest insertion of a dart by `I`. Second, `B, B_:fmap->dim->dart->fmap` break the latest `k`-link inserted for a dart by `L`, forward and backward respectively.

3.3. Hypermaps

Preconditions written as predicates are introduced for `I` and `L`:

```
Definition prec_I(m:fmap)(x:dart):= x <> nil /\ ~ exd m x.
Definition prec_L(m:fmap)(k:dim)(x y:dart):=
 exd m x /\ exd m y /\ ~ succ m k x /\ ~ pred m k y /\ cA m k x <> y.
```

If `I` and `L` are used under these conditions, the free map built necessarily has open orbits. In fact, thanks to the closures `cA` and `cA_1`, it can always be considered as a true hypermap exactly equipped with operations α_k of Definition 2.1. It satisfies the *invariant*:

```
Fixpoint inv_hmap(m:fmap):Prop:=
 match m with
    V => True | I m0 x => inv_hmap m0 /\ prec_I m0 x
  | L m0 k0 x y => inv_hmap m0 /\ prec_L m0 k0 x y
 end.
```

Such a hypermap was already drawn in Fig. 3. Fundamental proven properties are that, for any `m` and `k`, `(A m k)` and `(A_1 m k)` are *injections* inverse of each other, and `(cA m k)` and `(cA_1 m k)` are *permutations* inverse of each other, and are closures. Finally, traversals of faces are based on function `F` and its closure `cF`, which correspond to ϕ (Definition 2.2). So, in Fig. 3, `F m 1 = nil, cF m 1 = 5`. Properties similar to the ones of `A`, `cA` are proven for `F`, `cF` and their inverses `F_1`, `cF_1`.

3.4. Orbits

Testing if there exists a path from a dart to another in an orbit for a hypermap permutation is of prime importance, for instance to determine the number of orbits. The problem is exactly the same for α_0, α_1 or ϕ (Definitions 2.1 and 2.2). That is why a *signature* `Sigf` with formal parameters `f`, `f_1` and their properties is first defined.

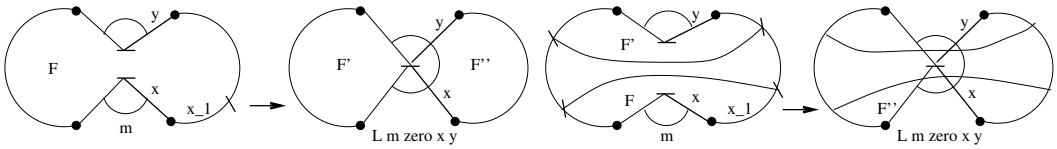


Figure 4: Linking at dimension 0.

Next, a *generic module* (or *functor*) `Mf(M:Sigf)`, the formal parameter `M` being a *module* of type `Sigf`, is written in Coq to *package* generic definitions and proven properties about `f` and `f_1`. Among them, we have that each `f`-orbit of `m` is *periodic* with a positive smallest uniform period for any dart `z` of the orbit. The predicate `expo m z t` asserts the *existence of a path* in an `f`-orbit of `m` from a dart `z` to another `t`, which is proven to be a *decidable equivalence*. Note that most of the properties are obtained by *noetherian induction* on the length of iterated sequences of `f`-successors, bounded by the period.

Appropriate modules, called `MA0`, `MA1` and `MF`, are written to instantiate for `(cA m zero)`, `(cA m one)` and `(cF m)` definitions and properties of `f`. So, a generic definition or property in `Mf(M)` has to be prefixed by the module name to be concretely applied. For instance, `MF.expo m z t` is the existence of a path from `z` to `t` in a face. In the following, `MF.expo` is abbreviated into `expf`. For instance, in Fig. 3, `expf m 1 5 = True, expf m 5 3 = False`. Finally, a binary relation `eqc` stating that two darts belong to the same component is easily defined by induction. For instance, in Fig. 3, we have `eqc m 1 5 = True, eqc m 1 13 = False`. We quickly prove that `(eqc m)` is a *decidable equivalence*.

3.5. Characteristics, Genus Theorem and Euler Formula

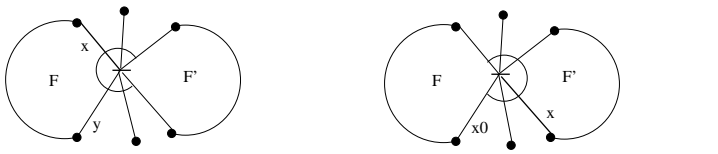
We now count cells and components of a hypermap using the Coq library module `ZArith` containing all the features of `Z`, the integer ring, including tools to solve linear systems in Presburger's arithmetics. The numbers `nd`, `ne`, `nv`, `nf` and `nc` of darts, edges, vertices, faces and components are easily defined by induction. Euler's characteristic `ec` and `genus` derive. The Genus Theorem and the Euler Formula (for any number `(nc m)` of components) are obtained as corollaries of the fact that `ec` is even and satisfies `2 * (nc m) >= (ec m)` (8). Remark that `->` denotes a functional type in `Set` as well as an implication in `Prop`:

```
Definition ec(m:fmap): Z:= nv m + ne m + nf m - nd m.
Definition genus(m:fmap): Z:= (nc m) - (ec m)/2.
Definition planar(m:fmap): Prop:= genus m = 0.
Theorem Genus_Theorem: forall m:fmap, inv_hmap m -> genus m >= 0.
Theorem Euler_Formula: forall m:fmap, inv_hmap m -> planar m ->
     ec m / 2 = nc m.
```

4. Planarity and connectivity criteria

A consequence of the previous theorems is a completely constructive *criterion of planarity*, when one correctly *links* with `L` at dimensions 0 or 1, *e.g.* for 0:

```
Theorem planarity_crit_0: forall (m:fmap)(x y:dart),
  inv_hmap m -> prec_L m zero x y -> (planar (L m zero x y) <->
    (planar m /\ (~ eqc m x y \/ expf m (cA_1 m one x) y))).
```



a. Double−link coded by (x, true) : face F contains y. b. Double−link coded by (x, false) : face F contains x0.

Figure 5: Coding a double-link and identifying a face.

So, the planarity of `m` is preserved for `(L m zero x y)` *iff* one of the following two conditions holds: (1) `x` and `y` are not in the same component of `m`; (2) `x_1 = (cA_1 m one x)` and `y` are in the same face of `m`, *i.e.* the linking operates *inside the face* containing `y`. Fig. 4 illustrates 0-linking inside a face, giving two new faces, and between two (connected) faces, giving a new face, thus destroying planarity. Finally, after a long development, we prove the expected *planarity criterion*, when breaking a link with `B`, at any dimension, *e.g.* for 0:

```
Lemma planarity_crit_B0: forall (m:fmap)(x:dart), inv_hmap m ->
   succ m zero x -> let m0 := B m zero x in let y := A m zero x  in
(planar m <-> (planar m0 /\ (~ eqc m0 x y \/ expf m0 (cA_1 m0 one x) y))).
```

Such a lemma is easy to write/understand as a *mirror form* of the 0-linking criterion, but it is much more difficult to obtain. It would be fruitful to relate these constructive/destructive criteria with the static one of G. Gonthier (10). Finally, some useful results quickly characterize the effect of a link break on the *connectivity* of a planar hypermap. For instance, when 0-breaking `x`, a disconnection occurs *iff* `expf m y x0`:

```
Lemma disconnect_planar_criterion_B0:forall (m:fmap)(x:dart),
  inv_hmap m -> planar m -> succ m zero x ->
    let y := A m zero x in let x0 := bottom m zero x in
      (expf m y x0 <-> ~eqc (B m zero x) x y).
```

5. Rings of faces

5.1. Coding a double-links and identifying a face

Since an edge is always open in our specification, when doing the backward break of a unique 0-link from `y` or `y'`, we in fact realize a double-link break, as in Definition 2.8. So, we choose to identify a double-link by the unique dart, we called `x`, where the 0-link to be broken begins. In fact, with respect to the face `F` *on the left of* the double-link in the ring, there are two cases, depending on the position of `x` and its forward 0-link, as shown in Fig. 5 (a) and (b). We decided to distinguish them by a Boolean `b`. Then, a double-link is coded by a pair `(x, b)`. So, we implicitely identify each ring face `F` by the double-link coding on its right in the ring. In Fig. 5 (a), face `F` is identified by `(x, true)` and contains `y:= A m zero x`, whereas in Fig. 5 (b), face `F` is identified by `(x, false)` and contains `x0:= bottom m zero x`. These modeling choices considerably simplify the problems. Indeed, in closed orbits, a true double-link break would entail 2 applications of `B` followed by 2 applications of `L`, and would be much more complicated to deal with in proofs.

5.2. Modeling a ring of faces

First, we inductively define linear lists of pairs of booleans and darts, with the two classical constructors `lam` and `cons`, and usual observers and destructors, which we do not give, because their effect is directly comprehensible:

```
Inductive list:Set := lam: list | cons: dart*bool -> list -> list.
```

Such a list is composed of couples `(x, b)`, each identifying a face `F`: if `b` is `true`, `F` is represented by `y:= A m zero x`, otherwise by `x0:= bottom m zero x` (Fig. 5). In the following, `Bl m l` breaks all the 0-links starting from the darts of list `l` in a hypermap `m`. Now, we have to model the conditions required for list `l` to be a ring of hypermap `m`. Translating Definition 2.7, we have four conditions, called `pre_ringk m l`, for $k = 0, \ldots, 3$, which we explain in the following sections. Finally, a predicate `ring` is defined by:

```
Definition ring(m:fmap)(l:list):Prop:= ~emptyl l /\
  pre_ring0 m l /\ pre_ring1 m l /\ pre_ring2 m l /\ pre_ring3 m l.
```

5.3. Ring Condition (0): unicity

The predicate `distinct_edge_list m x l0` saying that the edges of `l0` are distinct in `m` from a given edge of `x`, `pre_ring0 m l` is defined recursively on `l` to impose that all edges in `l` are distinct: Condition (0) of Definition 2.7. It also imposes that each dart in `l` has a 0-successor, in order to have well defined links, which is implicit in the mathematical definition, but not in our specification whith open orbits.

```
Fixpoint pre_ring0(m:fmap)(l:list){struct l}:Prop:=
 match l with
   lam => True | cons (x,_) l0 =>
     pre_ring0 m l0 /\ distinct_edge_list m x l0 /\ succ m zero x
 end.
```

5.4. Ring Condition (1): continuity

Then, we define adjacency between two faces identified by `xb = (x, b)` and `xb' = (x', b')`, along the link corresponding to `xb`:

```
Definition adjacent_faces(m:fmap)(xb xb':dart*bool):=
 match xb with (x,b) => match xb' with (x',b') =>
 let y := A m zero x in let y':= A m zero x' in
 let x0 := bottom m zero x in let x'0:= bottom m zero x' in
   if eq_bool_dec b true
   then if eq_bool_dec b' true then expf m x0 y' else expf m x0 x'0
   else if eq_bool_dec b' true then expf m y y' else expf m y x'0
 end end.
```

This definition is illustrated in Fig. 6 for the four possible cases of double-link codings. So, the predicate `pre_ring1 m l` recursively specifies that two successive faces in `l` are adjacent: Condition (1) in Definition 2.7:

```
Fixpoint pre_ring1(m:fmap)(l:list){struct l}:Prop:=
 match l with
  lam => True | cons xb l0 => pre_ring1 m l0 /\
    match l0 with lam => True | cons xb' l' => adjacent_faces m xb xb' end
 end.
```

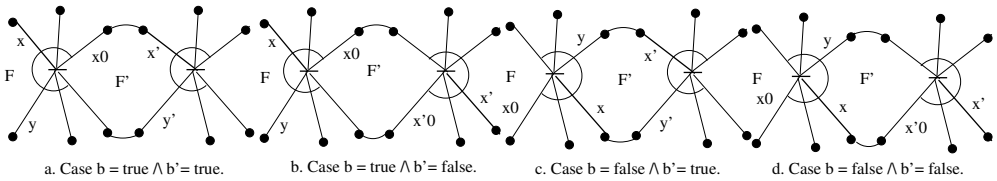


Figure 6: Four cases of face adjacency.

5.5. Ring Condition (2): circularity, or closure

The predicate `pre_ring2 m l` specifies that the last and first faces in `l` are adjacent: Condition (2) of circularity in Definition 2.7:

```
Definition pre_ring2(m:fmap)(l:list):Prop:=
 match l with
    lam => True | cons xb l0 =>
     match xb with (x,b) => let y := A m zero x in match l0 with
       lam => let x0 := bottom m zero x in expf m y x0
     | cons _ l' => let xb':= last l0 in adjacent_faces m xb' xb
     end end
end.
```

5.6. Ring Condition (3): simplicity

The predicate specifiying that the faces of `m` identified by `xb` and `xb'` are distinct is easy to write by cases on the Booleans in `xb` and `xb'`. The predicate `distinct_face_list m xb l0` expressing that the face identified by `xb` is distinct from all faces of list `l0` entails. Then, the predicate `pre_ring3 m l` says that all faces of `l` are distinct: Condition (3) in Definition 2.7:

```
Fixpoint pre_ring3(m:fmap)(l:list){struct l}:Prop:=
 match l with
  lam => True | cons xb l0 => pre_ring3 m l0 /\ distinct_face_list m xb l0
 end.
```

6. Discrete Jordan Curve Theorem

The general principle of the JCT proof for a hypermap `m` and a ring `l` is a structural induction on `l`. The case where `l` is empty is immediatly excluded because `l` is not a ring by definition. Thus the true first case is when `l` is *reduced to one element*, *i.e.* is of the form `cons (x, b) lam`. Then, we prove the following lemma as a direct consequence of the planarity criterion `planarity_crit_B0` and the criterion `face_cut_join_criterion_B0`:

```
Lemma Jordan1:forall(m:fmap)(x:dart)(b:bool), inv_hmap m -> planar m ->
   let l:= cons (x,b) lam in ring m l -> nc (Bl m l) = nc m + 1.
```

When a ring `l1` contains *at least two elements*, we prove that the condition `~expf m y x0` must hold with the first element `(x,b)` of `l1` (in fact, conditions (1) and (3) are enough):

```
Lemma ring1_ring3_connect:
 forall(m:fmap)(x x':dart)(b b':bool)(l:list), inv_hmap m ->
 let l1:= cons (x,b) (cons (x',b') l) in
 let y:=A m zero x in let x0:= bottom m zero x in
   planar m -> pre_ring1 m l1 -> pre_ring3 m l1 -> ~expf m y x0.
```

In this case, thanks to `disconnect_planar_criterion_B0` (Section 4), the lemma entails that the break of the first ring link does never disconnect the hypermap. Then, after examining the behavior of `pre_ringk`, for `k` $= 0, \ldots, 3$, we are able to prove the following lemma which states that the four ring properties are preserved after the first break in `l`:

```
Lemma pre_ring_B: forall(m:fmap)(l:list), inv_hmap m -> planar m ->
 let x := fst (first l) in let y := A m zero x in
 let x0 := bottom m zero x in let m1 := B m zero x in
~expf m y x0 -> ring m l -> (pre_ring0 m1 (tail l) /\ pre_ring1 m1 (tail l)
    /\ pre_ring2 m1 (tail l) /\ pre_ring3 m1 (tail l)).
```

The most difficult is to prove the part of the result concerning `pre_ringk`, for `k` $= 0, \ldots, 3$. The four proofs are led by induction on `l` in separate lemmas. For `pre_ring0`, the proof is rather simple. But, for the other three, the core is a long reasoning where 2, 3 or 4 links are involved in input. Since each link contains a Boolean, sometimes appearing also in output, until $2^4 = 16$ cases are to be considered to combine the Boolean values.

Finally, from `Jordan1` and `pre_ring_B` above, we have the expected result by a quick reasoning by induction on `l`, where links are broken one by one from the first:

```
Theorem Jordan: forall(l:list)(m:fmap),
  inv_hmap m -> planar m -> ring m l -> nc (Bl m l) = nc m + 1.
```

It is clear that, provided a mathematical hypermap M and a mathematical ring R conform to Definitions 2.1 and 2.7, we can always describe them as terms of our specification framework in order to apply our JCT. Conversely, given a hypermap term, some mathematical rings cannot directly be written as terms. To do it, our ring description and our JCT proof have to be slightly extended. However, that is not necessary for the *combinatorial maps* (where α_0 is an involution) terms, for which our ring specification and our JCT formalization are *complete*. This is more than enough to affirm the value of our results.

7. Conclusion

We have presented a new discrete statement of the JCT based on hypermaps and rings, and a formalized proof assisted by the Coq system. Our hypermap modeling with *open orbits* simplifies and precises most of known facts. It also allows to obtain some new results, particularly about hypermap construction/destruction, connection/disconnection and planarity. This work involves a substantial framework of hypermap specification, which is built *from scratch*, *i.e.* exempt from any proper axiom. It is basically the same as the one we have designed to develop geometric modelers via algebraic specifications (2). So, we know how to efficiently implement all the notions we formally deal with.

The Coq system turned out to be a precious auxiliary to guide and check all the process of specification and proof. The preexistent framework of hypermap specification represents about 15,000 lines of Coq, and the JCT development about 5,000 lines, including about 25 new definitions, and 400 lemmas and theorems. Note that all results about the dimension 0

were actually proven, but some planarity properties about dimension 1, which are perfectly symmetrical, have just been admitted. However, the JCT formal proof is complete.

So, we have a solid foundation to tackle any topological problem involving orientable surface subdivisions. Extensions are in 2D or 3D computational geometry and geometric modeling by introducing embeddings (6; 2), and computer imagery by dealing with pixels (7) or voxels.

References

[1] Bauer, G., Nipkow, T.: The 5 Colour Theorem in Isabelle/Isar. In *Theorem Proving in HOL Conf.* (2002). LNCS **2410**, Springer-Verlag, 67–82.

[2] Bertrand, Y., Dufourd, J.-F.: Algebraic specification of a 3D-modeler based on hypermaps. *Graphical Models and Image Processing* **56**:1 (1994), 29–60.

[3] The Coq Team Development-LogiCal Project: The Coq Proof Assistant Reference Manual - Version 8.1, INRIA, France (2007). `http://coq.inria.fr/doc/main.html`.

[4] Cori, R.: Un Code pour les Graphes Planaires et ses Applications. *Astérisque* **27** (1970), Société Math. de France.

[5] Dehlinger, C., Dufourd, J.-F.: Formalizing the trading theorem in Coq. *Theoretical Computer Science* **323** (2004), 399–442.

[6] Dufourd, J.-F., Puitg, F.: Fonctional specification and prototyping with combinatorial oriented maps. *Comp. Geometry - Th. and Appl.* **16** (2000), 129–156.

[7] Dufourd, J.-F.: Design and certification of a new optimal segmentation program with hypermaps. *Pattern Recognition* **40** (2007), 2974–2993.

[8] Dufourd, J.-F.: A hypermap framework for computer-aided proofs in surface subdivisions: Genus theorem and Euler's formula. In: *22nd ACM SAC* (2007), 757–761.

[9] Françon, J.: Discrete Combinatorial Surfaces. CVGIP : Graphical Models and Image Processing **57**:1, (1995), 20–26.

[10] Gonthier, G.: A computer-checked proof of the Four Colour Theorem. Microsoft Research, Cambridge, `http://coq.inria.fr/doc/main.html` (2005), 57 pages.

[11] Griffiths, H.: *Surfaces.* Cambridge University Press (1981).

[12] Hales, T.: A verified proof of the Jordan curve theorem. Seminar Talk. Dep. of Math., University of Toronto (2005), `http://www.math.pitt.edu/~thales`.

[13] Kornilowicz A.: Jordan Curve Theorem. In: *Formalized Mathematics* **13**:4 (2005), Univ. of Bialystock, 481–491.

[14] Rosenfeld, A.: Picture Languages - Formal Models for Picture Recognition. In: *Comp. Science and Appl. Math.* **series**. Academic Press, New-York (1979).

[15] Tutte, W.T.: Combinatorial oriented maps. *Can. J. Math.*, **XXXI**:5 (1979), 986–1004.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 265-276
www.stacs-conf.org

TRIMMING OF GRAPHS, WITH APPLICATION TO POINT LABELING

THOMAS ERLEBACH [1], TORBEN HAGERUP [2], KLAUS JANSEN [3], MORITZ MINZLAFF [4], AND ALEXANDER WOLFF [5]

[1] Department of Computer Science, University of Leicester, Leicester LE1 7RH, England.
E-mail address: t.erlebach@mcs.le.ac.uk

[2] Institut für Informatik, Universität Augsburg, 86135 Augsburg, Germany.
E-mail address: hagerup@informatik.uni-augsburg.de

[3] Institut für Informatik und Praktische Mathematik, Universität Kiel, 24098 Kiel, Germany.
E-mail address: kj@informatik.uni-kiel.de

[4] Institut für Mathematik, Technische Universität Berlin, 10623 Berlin, Germany.
E-mail address: minzlaff@math.tu-berlin.de

[5] Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven, the Netherlands.
URL: www.win.tue.nl/~awolff

Abstract. For $t, g > 0$, a vertex-weighted graph of total weight W is (t, g)-*trimmable* if it contains a vertex-induced subgraph of total weight at least $(1 - 1/t)W$ and with no simple path of more than g edges. A family of graphs is *trimmable* if for each constant $t > 0$, there is a constant $g = g(t)$ such that every vertex-weighted graph in the family is (t, g)-trimmable. We show that every family of graphs of bounded domino treewidth is trimmable. This implies that every family of graphs of bounded degree is trimmable if the graphs in the family have bounded treewidth or are planar. Based on this result, we derive a polynomial-time approximation scheme for the problem of labeling weighted points with nonoverlapping sliding labels of unit height and given lengths so as to maximize the total weight of the labeled points. This settles one of the last major open questions in the theory of map labeling.

1998 ACM Subject Classification: G.2.2 Graph Theory, I.1.2 Algorithms.

Key words and phrases: Trimming weighted graphs, domino treewidth, planar graphs, point-feature label placement, map labeling, polynomial-time approximation schemes.

Work supported by grant WO 758/4-2 of the German Research Foundation (DFG).

1. Introduction

1.1. Graph Trimming

In this paper we investigate the problem of deleting vertices from a given graph so as to ensure that all simple paths in the remaining graph are short. We assume that each vertex has a nonnegative weight, and we want to delete vertices of small total weight. Whereas there is an extensive literature on separators, which can be viewed as serving to destroy all large connected components, we are not aware of previous work on vertex sets that destroy all long simple paths. Let us make our notions precise.

Definition 1.1. For $t > 0$ and $g \geq 0$, a (t, g)-*trimming* of a vertex-weighted graph $G = (V, E)$ of total weight W is a set $U \subseteq V$ of weight at most W/t such that every simple path in G of more than g edges contains a vertex in U. If G has a (t, g)-trimming, we also say that G is (t, g)-*trimmable*.

We say that a family of graphs is *trimmable* if, for every constant $t > 0$, there is a constant $g \geq 0$ (that depends only on t) such that every vertex-weighted graph in the family is (t, g)-trimmable. Of course, it suffices to demonstrate this for t larger than an arbitrary constant. Not every family of graphs is trimmable. For example, if $n, t \geq 2$ and we delete a $(1/t)$-fraction of the vertices in an unweighted n-clique K_n, the remaining graph still has a simple path of $n(1-1/t)-1$ edges. This expression is not bounded by a function of t alone, so the family of complete graphs is not trimmable.

With a little effort, one can show the family of trees to be trimmable. One popular generalization of trees is based on the definition below. Given a graph $G = (V, E)$ and a set $U \subseteq V$, we denote by $G[U]$ the subgraph of G induced by U. The *union* of graphs $G_i = (V_i, E_i)$, for $i = 1, \ldots, m$, is the graph $\bigcup_{i=1}^{m} G_i = (\bigcup_{i=1}^{m} V_i, \bigcup_{i=1}^{m} E_i)$.

Definition 1.2. A *tree decomposition* of an undirected graph $G = (V, E)$ is a pair (T, B), where $T = (X, E_T)$ is a tree and $B : X \to 2^V$ maps each node x of T to a subset of V, called the *bag* of x, such that

- $\bigcup_{x \in X} G[B(x)] = G$, and
- for all $x, y, z \in X$, if y is on the path from x to z in T, then $B(x) \cap B(z) \subseteq B(y)$.

The *width* of the tree decomposition (T, B) is $\max_{x \in X} |B(x)| - 1$, and the *treewidth* of G is the smallest width of any tree decomposition of G.

This standard definition is given, e.g., by Bodlaender [Bod98]. The family of graphs of treewidth at most 1 coincides with the family of forests. By analogy with several other generalizations from the family of trees to families of graphs of bounded treewidth, it seems natural to ask whether every family of graphs of bounded treewidth is trimmable. At present we cannot answer this question; we need a concept stronger than bounded treewidth alone.

Definition 1.3. The *elongation* of a tree decomposition (T, B) is the maximum number of edges on a simple path in T between two nodes with intersecting bags. For every $s \geq 0$, let the s-*elongation treewidth* of an undirected graph G be the smallest width of a tree decomposition of G with elongation at most s.

Since every graph has a trivial tree decomposition of elongation 0, the s-elongation treewidth of every graph is well-defined for every $s \geq 0$. The 1-elongation treewidth is the *domino treewidth* studied, e.g., by Bodlaender [Bod99].

Our main result about graph trimming, proved in Section 2, is that for all fixed $s \geq 0$, every family of graphs of bounded s-elongation treewidth is trimmable. Ding and Oporowski [DO95] proved that the domino treewidth of a graph can be bounded by a function of its usual treewidth and its maximum degree. It follows that every family of graphs of bounded treewidth and bounded degree is also trimmable. We derive from this that all families of planar graphs of bounded degree are trimmable as well. This result has applications described below.

1.2. Label Placement

Our main motivation for investigating trimmable graph families arose in the context of labeling maps with sliding labels. Generally speaking, map labeling is the problem of placing a set of labels, each in the vicinity of the object that it labels, while meeting certain conditions. For an overview, see the map-labeling bibliography [WS96]. First of all, labels are not allowed to overlap. As a consequence, it may not be possible to label all objects in a map, and the goal is to make an optimal selection according to some criterion. When a point feature such as a town or a mountain top is to be labeled, the label can usually be approximated without much loss by an axes-parallel rectangular shape and must be placed in the plane without rotation so that its boundary touches the point. One distinguishes between *fixed-position models* and *slider models*. In fixed-position models, each label has a predetermined finite set of *anchor points* on its boundary (e.g., the four corner points), and the label must be placed so that one of its anchor points coincides with the point to be labeled. In slider models, the anchor points form *anchor segments* on the boundary of the label (e.g., its bottom edge).

Van Kreveld et al. [vKSW99] introduced a taxonomy of fixed-position and slider models, which was later refined by Poon et al. [PSS$^+$03]. We use the slider models 1SH, 2SH and 4S of Poon et al., which define the anchor segments of a label to be its bottom edge, its top and bottom edges, and its entire boundary, respectively. We always require labels to be unit-height rectangles. This models the case in which all labels contain single text lines of the same character height. Fig. 1 illustrates the 1SH model. We assume that each point to be labeled comes equipped with a nonnegative weight, which may be used to express priorities among the points. If points represent villages, towns and cities on a map, priorities may correspond to the number of inhabitants, for example. Our objective is to label points with nonoverlapping labels so as to maximize the sum of the weights of those points that actually receive a label. This objective function causes points with large weights (e.g., large cities) to be likely to be labeled. We refer to the specific map-labeling problems described in this paragraph as *weighted unit-height 1SH-labeling*, etc. Since the qualifiers "weighted" and "unit-height" apply throughout the paper, we may occasionally omit them.

Recall that for $\rho \leq 1$, a *ρ-approximation algorithm* for a maximization problem is an algorithm that always outputs a solution of value at least ρ times the optimal objective value. An algorithm that takes an additional parameter $\varepsilon > 0$ and, for each fixed ε, is a polynomial-time $(1-\varepsilon)$-approximation algorithm is called a *polynomial-time approximation scheme* (*PTAS*). If the running time depends polynomially on ε as well, the algorithm is a *fully polynomial-time approximation scheme* (*FPTAS*).

Poon et al. [PSS$^+$03] show that finding an optimal weighted unit-height 1SH-labeling is NP-hard, even if all points lie on a horizontal line and the weight of each point equals the length of its label. For the one-dimensional case, in which all points lie on a horizontal

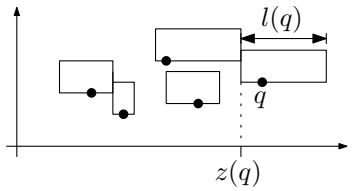


Figure 1: A 1SH-labeling L

line, they give an FPTAS, which yields an $O(n^2/\varepsilon)$-time $(1/2-\varepsilon)$-approximation algorithm for the two-dimensional case for arbitrary $\varepsilon > 0$. Poon et al. also describe a PTAS for unit-square labels. They raise the question of whether a PTAS exists for rectangular labels of arbitrary length and unit height. This is known to be the case for fixed-position models [AvKS98] and for sliding labels of unit weight [vKSW99]. The corresponding $(1-\varepsilon)$-approximation algorithms run in $n^{O(1/\varepsilon)}$ and $n^{O(1/\varepsilon^2)}$ time, respectively, for arbitrary $\varepsilon > 0$. The question of whether the combination of both sliding labels and arbitrary weights allows a PTAS has been one of the last major open problems in (theoretical point-feature) map labeling. In a preliminary version of this paper [EHJ$^+$06], we made some progress in answering this question. We gave a $(2/3-\varepsilon)$-approximation for weighted unit-height 1SH-labeling with running time $n^{O(1/\varepsilon^2)}$, for arbitrary $\varepsilon > 0$, and showed that the same approach yields a PTAS if the ratio of longest to shortest label length is bounded.

In Section 3 we settle the open question of Poon et al. by presenting a PTAS for weighted unit-height 1SH-labeling. There are no restrictions on label weights and lengths. Our approach is to discretize a given instance I of the weighted unit-height 1SH-labeling problem, i.e., to turn it into a fixed-position instance I', after which we can apply a known fixed-position algorithm to I'. The main difficulty is to find a "suitable" set of discrete label positions for each point. "Suitable" means that the weight of an optimal labeling of I' must be close enough to the weight of an optimal labeling of I. Dependencies between labels can be modeled via a graph, and long simple paths in this graph translate into large sets of anchor points that cannot be left out of consideration. Here our results from Section 2 come into play. We prove that the family of dependency graphs, if carefully defined, is trimmable, and we show how this may be used to bound the number of anchor points by a polynomial. We also show how to extend our PTAS for (weighted unit-weight) 1SH-labeling to the related 2SH-labeling and 4S-labeling problems.

2. Trimming of Graphs

In this section we show that for every constant s, every family of graphs of bounded s-elongation treewidth is trimmable. This implies that every family of graphs of bounded degree is trimmable if the graphs in the family have bounded treewidth or are planar.

Theorem 2.1. *Let $k, s \geq 0$ and suppose that a vertex-weighted undirected graph G has a tree decomposition D of width k and elongation s. Take $a = k+1$ if $s \geq 2$ and $a = \lceil k/2 \rceil$ if $s \leq 1$. Then, for every integer $t \geq 2$, G has a (t, g)-trimming, where $g = (2(s+1)t-3)(k+1)$ if $a \leq 1$ and*

$$g = (a^{(s+1)t-2}(a+1) - 2)(k+1)/(a-1)$$

if $a \geq 2$. Therefore, for every constant s, every family of graphs of bounded s-elongation treewidth is trimmable.

Proof. Let $D = (T, B)$, root T at an arbitrary node and let U be the set of vertices in bags whose depth d in T satisfies $d \bmod (s+1)t = i$, with the integer i chosen to minimize the weight of U. We show that U is a (t, g)-trimming of G.

Let $G = (V, E)$ and denote the total weight of the vertices in V by W. Since each vertex in V occurs in bags on at most $s+1$ levels in T, the sum, over all levels, of the weight of the vertices occurring in bags on the level under consideration is at most $(s+1)W$. Therefore, by the choice of i, the weight of U is at most $(s+1)W/((s+1)t) = W/t$, as desired.

Let $\pi = (v_0, \ldots, v_m)$ be a simple path in G of $m \geq 1$ edges and, for $i = 1, \ldots, m$, choose a node x_i in T whose bag contains both v_{i-1} and v_i. Because T is connected, there is a path from x_i to x_{i+1} (or they coincide), for $i = 1, \ldots, m-1$, so π can be viewed as inducing a walk π' in T. The walk π' may visit a node x in T several times. However, each visit to x "uses" a vertex in $B(x)$ that cannot be reused later, so no node of T occurs more than $k+1$ times on π'. If $s \leq 1$, we can strengthen this statement as follows: For $i = 1, \ldots, m-1$, the nodes x_i and x_{i+1} must coincide or be adjacent, so each visit by π' to a node x "uses" two vertices in $B(x)$, rather than just one, and the number of such visits is bounded by $\lfloor (k+1)/2 \rfloor = \lceil k/2 \rceil$. In either case, therefore, the nodes on π' span a subtree T' of T in which no node has more than a children, except that the root may have $a+1$ children. The number of nodes at depth d in such a tree is bounded by $(a+1)a^{d-1}$, for all $d \geq 0$, and therefore the number of nodes at depth at most d is bounded by $2d+1$ if $a = 1$ and by $1 + (a+1)(a^d - 1)/(a-1) = ((a+1)a^d - 2)/(a-1)$ if $a \geq 2$.

Suppose that π contains no vertex in U. Then, by the choice of U, the depth of T' is at most $(s+1)t - 2$, and the number of nodes in T' is at most $2(s+1)t - 3$ if $a = 1$ and at most $(a^{(s+1)t-2}(a+1) - 2)/(a-1)$ if $a \geq 2$. Since each bag contains at most $k+1$ vertices, it follows that $m + 1 \leq (2(s+1)t - 3)(k+1)$ if $a = 1$ and that $m + 1 \leq (a^{(s+1)t-2}(a+1) - 2)(k+1)/(a-1)$ if $a \geq 2$. ∎

Corollary 2.2. *For all integers $k \geq 0$, $d \geq 1$ and $t \geq 2$, every vertex-weighted undirected graph of treewidth k with maximum degree d has a $(t, \lceil K/2 \rceil^{2t})$-trimming, where $K = (9k+7)d(d+1) - 1$. Hence, every family of graphs with bounded degree and bounded treewidth is trimmable.*

Proof. According to Bodlaender [Bod99, Theorem 3.1], every such graph has a domino tree decomposition of width at most K. Except in the trivial case $k = 0$, we have $K \geq 31$. By Theorem 2.1, used with $s = 1$, the graph has a (t, g)-trimming, where

$$g = (\lceil K/2 \rceil^{2t-2}(\lceil K/2 \rceil + 1) - 2)(K+1)/(\lceil K/2 \rceil - 1) \leq \lceil K/2 \rceil^{2t}.$$

∎

We can extend this result to planar graphs of bounded degree.

Corollary 2.3. *For all integers $d, t \geq 1$, every vertex-weighted undirected planar graph of maximum degree d has a $(t, \lceil K/2 \rceil^{4t})$-trimming, where $K = (54t - 29)d(d+1) - 1$. Hence every family of planar graphs of bounded degree is trimmable.*

Proof. Let $G = (V, E)$ be a planar graph with maximum degree d and denote the total weight of the vertices in V by W. We first follow the approach of Baker [Bak94] to obtain a $(2t-1)$-outerplanar subgraph of G by deleting vertices of total weight at most $W/(2t)$.

Consider an arbitrary planar embedding of G. Partition the vertices of G into layers by repeatedly deleting the vertices on the boundary of the outer face until no vertex remains. The vertices deleted in one iteration of this process form a layer. Number the layers $R_1, R_2, \ldots$ in the order of their deletion. For every $j \in \{0, 1, \ldots, 2t-1\}$, consider the set V_j of vertices in layers R_i with $i \bmod (2t) = j$, choose j such that the total weight of V_j is at most $W/(2t)$ and consider the subgraph H_j of G induced by $V \setminus V_j$.

H_j is $(2t-1)$-outerplanar and thus has treewidth at most $6t-4$ [Bod98, Theorem 83]. By Corollary 2.2, H_j has a $(2t, \lceil K/2 \rceil^{4t})$-trimming U. The set $V_j \cup U$ has weight at most $W/(2t) + W/(2t) = W/t$ and is therefore a $(t, \lceil K/2 \rceil^{4t})$-trimming of G. ∎

Remark 2.4. A better dependence of the bound in Corollary 2.3 on t can be achieved by deleting less than $1/(2t)$ of the weight of the graph in the first step, so that more than $1/(2t)$ of the weight can be deleted when Corollary 2.2 is applied. In this way, the treewidth of H_j and thus the value of K increases, but the exponent of the bound becomes smaller than $4t$. More precisely, if we delete $1/(\alpha t)$ of the weight in the first step, for some $\alpha > 2$, then the resulting bound is $\lceil K/2 \rceil^{2\lceil \alpha t/(\alpha-1) \rceil}$ with $K = (27\alpha t - 29)d(d+1) - 1$. For each pair (d, t), there is a value of α that optimizes the resulting bound.

3. Labeling Weighted Points with Sliding Labels

In this section we define the labeling problems of relevance to us formally and show that there are polynomial-time approximation schemes for weighted unit-height 1SH-labeling, 2SH-labeling and 4S-labeling. We use $\mathbb{R}$, $\mathbb{R}_{>0}$ and $\mathbb{R}_{\geq 0}$ to denote the sets of real numbers, of positive real numbers and of nonnegative real numbers, respectively, and $\mathbb{R}^2$ is the two-dimensional Euclidean plane.

Definition 3.1. An instance of the *weighted unit-height 1SH-labeling problem* is a triple $I = (P, l, w)$, where P is a finite subset of $\mathbb{R}^2$ and $l : P \to \mathbb{R}_{>0}$ and $w : P \to \mathbb{R}_{\geq 0}$ are functions defined on P. $|P|$ is called the *size* of I.

In the definition of 1SH-labeling, P represents the set of points to be labeled, and for each $p \in P$, $l(p)$ is the length of the label of p and $w(p)$ is the weight of p. When (P, l, w) is an instance of the 1SH-labeling problem and $Q \subseteq P$, we call $w(Q) = \sum_{p \in Q} w(p)$ the *weight* of Q.

Definition 3.2. A feasible solution or *labeling* of an instance $I = (P, l, w)$ of the weighted unit-height 1SH-labeling problem is a pair $L = (Q, z)$, where $Q \subseteq P$ and $z : Q \to \mathbb{R}$ is a function with $p_x - l(p) \leq z(p) \leq p_x$ for all $p = (p_x, p_y) \in Q$ such that for all $p = (p_x, p_y)$ and $q = (q_x, q_y)$ in Q with $p \neq q$ and $|p_y - q_y| < 1$, either $z(p) + l(p) \leq z(q)$ or $z(q) + l(q) \leq z(p)$. The *weight* of L is the weight of Q, and L is *optimal* if no labeling of I has greater weight than L.

Informally, Q is the set of points in P that receive a label, and for each $p \in Q$, $z(p)$ denotes the x-coordinate of the left edge of the label of p. The condition $p_x - l(p) \leq z(p) \leq p_x$ for all $p = (p_x, p_y) \in Q$ expresses that p lies on the bottom edge of its label. Let us say that two points $p = (p_x, p_y)$ and $q = (q_x, q_y)$ in $\mathbb{R}^2$ *y-overlap* if $|p_y - q_y| < 1$. The condition $z(p) + l(p) \leq z(q)$ or $z(q) + l(q) \leq z(p)$ for each pair (p, q) of distinct y-overlapping points in Q expresses that labels are not allowed to overlap.

We define an instance of the *weighted unit-height multi-position labeling* or *1MH-labeling problem* as a pair $(I, \mathcal{M})$, where $I = (P, l, w)$ is an instance of the weighted unit-height 1SH-labeling problem and $\mathcal{M}$ is a function that maps each point in P to a finite subset of $\mathbb{R}$. A *labeling* of $(I, \mathcal{M})$ is a labeling (Q, z) of I such that $z(p) \in \mathcal{M}(p)$ for all $p \in Q$. If $\mathcal{M}$ maps all $p \in P$ to the same set $M \subseteq \mathbb{R}$, we may write $(I, \mathcal{M})$ as (I, M). The principal technical contribution of this section is a reduction of 1SH-labeling to 1MH-labeling. Before giving a precise description of the reduction, we provide an informal overview.

The reduction maps an instance I of 1SH-labeling to an instance of 1MH-labeling of the form (I, M), where $M \subseteq \mathbb{R}$. It therefore suffices to show that a suitable set M exists and can be computed sufficiently fast. As a step towards this goal, we describe a *normalization* procedure that transforms an arbitrary given labeling of I into one of (I, M). The normalization is introduced for the sake of argument only and is not actually carried out as part of the reduction.

The top-level idea behind the normalization is to process the labels of the given labeling in the order from left to right, pushing each label as far to the left as it can go without bumping into another label or being separated from the point that it labels. It is easy to observe that in every normalized labeling, the position of each label (taken to be the x-coordinate of its left edge) is the sum of the x-coordinate of some labeled point and some number of label lengths, minus its own length. This still leaves too many possibilities, however, since essentially every selection of points to receive labels may give rise to a different position of a given label.

The dependencies between labels can be modeled in a natural way through a directed *dependency graph* G: If the label of a point q, moving left, may bump into that of a point p, then G includes the edge (p, q). The problem identified above stems from the fact that G may have very long paths, corresponding to chains of many labels that may touch and influence each other. Our defense against this is trimming, so we must ensure that G is trimmable. Assuming that this is so, we can break all paths with more than a constant number of edges by dropping labels of small total weight, which reduces the number of possible label positions to a polynomial. Afterwards we must re-normalize, however, since otherwise the trimming buys us nothing. This gives rise to another problem, in that the re-normalization may create new long paths. In order to counter this, we introduce vertical *stopping lines* and modify the normalization to never push the left edge of a label past a stopping line. As long as at least one stopping line passes through each dropped label (including its boundary), we can be sure that the re-normalization creates no new paths. Fairly arbitrarily, for every label, we choose to put stopping lines through the left and right edges of the area occupied by the label in its leftmost position (if no other labels obstruct its movement). This also ensures in a simple way that no label gets separated from the point that it labels. Now labels with their right edge to the left of or on a stopping line ℓ cannot influence labels with their left edge to the right of or on ℓ, so we can remove all edges from G that cross a stopping line. This turns out to have the beneficial effect of making G planar and of bounded degree, which implies that it is trimmable, as needed above.

By attaching real-valued lengths to the edges of G and adding an additional vertex O with incident edges described below to G, we can obtain the position of the label of each point p as the length of a path from O to p. Every edge (p, q) between two points p and q is given a length equal to that of the label of p, since that is the distance that the left edge of the label of q must keep from that of p. Every stopping line ℓ, passing through $(x, 0)$, say, and every point p give rise to an edge from O (which can be thought of as representing the

y-axis) to p of length x, since x is the distance that the left edge of the label of p, because of ℓ, must keep from the y-axis if it begins its movement to the right of ℓ or on ℓ. Now the label of each point p will move to a position that is precisely the largest length of a path from O to p no larger than the original position of the label.

Every stopping line adds to the number of possible label positions in a normalized labeling, but the dependence on the number of stopping lines is only linear. In fact, because of a later need for this added flexibility, Lemma 3.3 below allows the specification of an arbitrary set S of x-coordinates of additional stopping lines. The fact that the left edge of a label crosses no additional stopping line as it moves left can be expressed by saying that the movement leaves the rank in S of the position of the label invariant.

Lemma 3.3. *Given an instance $I = (P, w, l)$ of the weighted unit-height 1SH-labeling problem of size n, a finite set $S \subseteq \mathbb{R}$ and an $\varepsilon \in \mathbb{R}$ with $0 < \varepsilon \leq 1$, in $O((n + |S|)n^g)$ time, where $g = (1/\varepsilon)^{O(1/\varepsilon)}$, we can compute a set $M \subseteq \mathbb{R}$ with $|M| \leq (2n + |S|)n^g$ such that for every labeling (Q, z) of I, the instance (I, M) of the weighted unit-height 1MH-labeling problem has a labeling (Q', z') with $Q' \subseteq Q$ of weight at least $(1 - \varepsilon)w(Q)$ such that for all $p \in Q'$, $z'(p) \leq z(p)$ and $z'(p)$ and $z(p)$ have the same rank in S.*

Proof. Take $S' = S \cup \bigcup_{(p_x,p_y)\in P}\{p_x - l(p), p_x\}$ and let $G = (Q, E)$ be the directed graph with edge lengths on the vertex set Q that, for all $p = (p_x, p_y)$ and $q = (q_x, q_y)$ in Q, contains the edge (p, q) with length $l(p)$ exactly if $p_x < q_x$, $|p_y - q_y| < 1$ and there is no $x \in S'$ with $z(p) + l(p) \leq x \leq z(q)$. Moreover, let H be the undirected graph on the vertex set Q that contains an edge $\{p, q\}$, for all $p, q \in Q$ with $p \neq q$, exactly if p and q y-overlap.

Let us say that two points $p = (p_x, p_y)$ and $r = (r_x, r_y)$ in Q *x-surround* a point $q = (q_x, q_y)$ if $p_x \leq q_x \leq r_x$ or $r_x \leq q_x \leq p_x$. Let p, $q = (q_x, q_y)$ and r be three points in Q, every two of which y-overlap, and suppose that $z(p) \leq z(q) \leq z(r)$. Then we must clearly have $z(p) + l(p) \leq z(q) \leq q_x \leq z(q) + l(q) \leq z(r)$, which, since $q_x \in S'$, implies that $(p, r) \notin E$. This proves the following *triangle property*: If $(p, q) \in E$, then p and q x-surround no neighbor of both in H.

If $p = (p_x, p_y) \in Q$, then all in- and out-neighbors of p in G lie in the open horizontal strip of height 2 centered on the line $y = y_p$. Therefore, if p has in- or out-degree 3 or more, two in-neighbors or two out-neighbors of p are neighbors in H, which contradicts the triangle property. Thus all in- and out-degrees of G are bounded by 2.

We next prove that G is planar. Consider an embedding of G that maps each point in Q to itself and each edge in E to a straight line segment and assume to the contrary that for two edges (p_1, q_1) and (p_2, q_2) in E with $|\{p_1, q_1, p_2, q_2\}| = 4$, the corresponding closed line segments $\overline{p_1q_1}$ and $\overline{p_2q_2}$ intersect in a point $u = (u_x, u_y)$. Call p_1 and q_1 as well as p_2 and q_2 *partners* and let H_4 be the subgraph of H spanned by the vertex set $Q_4 = \{p_1, q_1, p_2, q_2\}$.

All points in Q_4 lie in the open horizontal strip of height 2 centered on the line ℓ defined by $y = u_y$. If there are a topmost point in Q_4 (one of maximal y-coordinate) and a bottommost point in Q_4 that are partners, then, since these y-overlap, all pairs of points in Q_4 y-overlap, and H_4 is a complete graph. Otherwise there is a unique topmost point and a unique bottommost point in Q_4, these *extreme* points are not partners, and each of the two other points in Q_4 lies on ℓ or on the opposite side of ℓ with respect to its extreme partner. Each nonextreme point in Q_4 y-overlaps both extreme points, and hence also the fourth point in Q_4, either by virtue of lying on ℓ or because one extreme point is its partner, while the other extreme point lies on the same side of ℓ as itself. This means that H_4 is a complete graph, except that the two extreme points may not be neighbors.

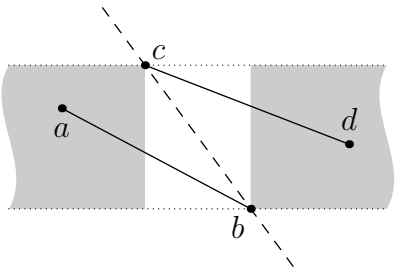


Figure 2: a and d lie in distinct gray areas and are therefore on opposite sides of $\overline{bc}$.

Because the two line segments between partners intersect, some two points in Q_4 that are partners, say, a and b, must x-surround another point in Q_4, say, c. By the triangle property, H lacks one of the edges $\{a, c\}$ and $\{b, c\}$, say, $\{b, c\}$, so H is not complete and b and c are extreme. The partner of c, say, d, is not extreme, so it is not x-surrounded by a and b. This implies that c and d x-surround a or b and, in fact, since a is not extreme, that they x-surround b. The two extreme points b and c can now be seen to be x-surrounded by a and d. But then it is geometrically clear that a and d belong to opposite open halfspaces bounded by the line through b and c (see Fig. 2), a contradiction to the fact that $\overline{ab}$ and $\overline{cd}$ intersect.

We have demonstrated that G is planar and of bounded degree and therefore trimmable. With $t = 2/\varepsilon$, let U be a (t, g)-trimming set of G for some integer $g \geq 0$ with $g = t^{O(t)}$—this is possible by Corollary 2.3—and take $Q' = Q \setminus U$. Let $\overline{G}$ be the multigraph obtained from G by adding a new vertex O and, for each $x \in S'$ and each $p \in Q$, an edge from O to p of length x.

For all $p \in Q'$, let a *p-path* be a path in $\overline{G}[\{O\} \cup Q']$ from O to p and define the *length* of a p-path as the sum of the lengths of its edges. For all $p = (p_x, p_y) \in Q'$, let $z'(p)$ be the largest length of a p-path that does not exceed $z(p)$—this is well-defined since $z(p) \geq p_x - l(p)$, while there is an edge, and hence a path, in $\overline{G}$ from O to p of length $p_x - l(p)$. We will show that (Q', z') is a labeling of I. First, for each $p = (p_x, p_y) \in Q'$, the relation $p_x - l(p) \leq z'(p) \leq z(p) \leq p_x$ was essentially argued above. Second, we must show, informally speaking, that the labels of the points in Q', if placed as indicated by z', do not overlap.

Let $p = (p_x, p_y)$ and $q = (q_x, q_y)$ be y-overlapping points in Q' and assume, without loss of generality, that $z(p) \leq z(q)$ and therefore that $z(p) + l(p) \leq z(q)$. If G contains the edge (p, q), then, since $z'(p)$ is the length of a p-path, $z'(p) + l(p)$ is the length of a q-path and, by definition of z', we have $z'(q) \geq z'(p) + l(p)$. If G does not contain the edge (p, q), there is an $x \in S'$ with $z(p) + l(p) \leq x \leq z(q)$. Again by definition of z', since $\overline{G}$ contains an edge from O to q of length x, it follows that $z'(q) \geq x \geq z(p) + l(p) \geq z'(p) + l(p)$. In either case, the labels of p and q, placed according to z', do not overlap.

We have $w(Q') \geq (1 - 1/t)w(Q)$, and for each $p \in Q'$, $z'(p)$ is the length of a p-path. The length of every p-path belongs to the set M of all sums of an element of S' and at most g elements of $\{l(p) \mid p \in P\}$. The set M is of size at most $(2n + |S|)n^g$ and can be computed in $O((n + |S|)n^g)$ time. Let $p \in Q'$. Since for each $x \in S$ there is a p-path of length x, it is easy to see that stepping from $z(p)$ to $z'(p)$ does not descend strictly below any $x \in S$, i.e., $z'(p)$ has the same rank in S as $z(p)$. ∎

We need to show how to solve the instance of the 1MH-labeling problem obtained using Lemma 3.3. Agarwal et al. [AvKS98] have given a PTAS that finds near-maximum independent sets in any given set of axes-aligned unit-height rectangles. They assume that rectangles are topologically closed. Under this assumption it is easy to argue that their PTAS for maximum independent set at the same time is a PTAS for maximizing the number of points labeled with unit-height rectangular labels in some fixed-position model. The reason is simply that, by definition, any two label candidates of the same point must touch this point. If label candidates are closed, one label candidate automatically excludes the other from the solution. Unfortunately, this is not the case if we consider labels to be open; e.g., in the 1SH-model the leftmost and the rightmost label candidate of a point do *not* intersect, so an algorithm for maximum independent set would not automatically yield feasible solutions for multi-position labeling. However, we can adapt the PTAS of Agarwal et al. to this case. In fact, the adapted PTAS can deal with the *weighted unit-height generalized multi-position labeling* or *4M-labeling problem*, in which each label specifies an arbitrary finite set of anchor points on its boundary. If a point is labeled, its label must be placed so that one of its anchor points coincides with the point to be labeled.

Lemma 3.4. *There is a PTAS for the weighted unit-height 4M-labeling problem. The running time for computing a* $(1-\varepsilon)$*-approximate solution is* $n^{O(1/\varepsilon)}$*, for all* ε *with* $0 < \varepsilon \leq 1$*.*

Clearly, a PTAS for 4M-labeling is also a PTAS for the more restricted 1MH-labeling problem.

Theorem 3.5. *Given an instance* I *of the weighted unit-height 1SH-labeling problem of size* n *and an* $\varepsilon \in \mathbb{R}$ *with* $0 < \varepsilon \leq 1$*, a labeling of* I *of weight at least* $(1-\varepsilon)$ *times the weight of an optimal labeling of* I *can be computed in* $n^{t^{O(t)}}$ *time, where* $t = 2/\varepsilon$*. The weighted unit-height 1SH-labeling problem therefore admits a PTAS.*

Proof. Let W^* be the weight of an optimal labeling of I. Use the algorithm of Lemma 3.3 with $S = \emptyset$ to compute a set $M \subseteq \mathbb{R}$ with $|M| \leq 2n^{g+1}$, where $g = t^{O(t)}$, such that the instance $I' = (I, M)$ of the weighted unit-height 1MH-labeling problem has a labeling of weight at least $(1-1/t)W^*$. Applying the PTAS of Lemma 3.4 to I', we obtain a labeling of I', and therefore of I, of weight at least $(1-1/t)^2 W^* \geq (1-2/t)W^* = (1-\varepsilon)W^*$ in time $(n^{g+2})^{O(t)} = n^{t^{O(t)}}$, which dominates the time needed by the first step. ∎

This result can be extended without much effort to the slightly more general labeling model 2SH, where a label must touch the point labeled with either its top or bottom edge.

Corollary 3.6. *There is a PTAS for weighted unit-height 2SH-labeling.*

Proof. 2SH-labeling can be reduced to 1SH-labeling—imagine adding to each original input point a copy at a distance of 1 below it. Then we use the reduction from 1SH-labeling to 1MH-labeling described in Lemma 3.3. In the resulting instance of 1MH-labeling, we discard the copies of points and view each label of a copy of a point as labeling the original point. Now we can apply the PTAS of Lemma 3.4 to the resulting instance of 4M-labeling. ∎

A further generalization allows us to deal also with the most general slider model, 4S, in which a label may have the point that it labels anywhere on its boundary.

Corollary 3.7. *There is a PTAS for weighted unit-height 4S-labeling.*

Proof sketch. Let an instance $I = (P, l, w)$ of the 4S-labeling problem (which is the same as an instance of the 1SH-labeling problem) be given. Each point $p \in P$ can be labeled with a horizontally sliding label that touches p with its bottom edge (or top edge), or by a vertically sliding label that touches p with its left edge (or right edge). This means that there are four types of rectangles that can potentially label p, all of which are taken into account in the following. Applying Lemma 3.3 twice (once horizontally and once vertically), we compute an instance I_{h} of the 1MH-labeling problem for the positions of horizontally sliding labels, specifying vertical stopping lines at x-positions $p_x - l(p)$, p_x and $p_x + l(p)$ for all $p = (p_x, p_y)$ in P, and another instance I_{v} for the positions of vertically sliding labels, specifying horizontal stopping lines at y-positions $p_y - 1$, p_y and $p_y + 1$ for all $p = (p_x, p_y)$ in P. Consider an optimal labeling L of I and let Q be the set of points that it labels. Let Q_{h} and Q_{v} be the sets of points in Q that are labeled with a horizontally sliding label and with a vertically sliding label, respectively. By Lemma 3.3, there is a solution L'_{h} for I_{h} that labels points $Q'_{\mathrm{h}} \subseteq Q_{\mathrm{h}}$, and a solution L'_{v} for I_{v} that labels points $Q'_{\mathrm{v}} \subseteq Q_{\mathrm{v}}$, of weights at least $(1-\varepsilon)w(Q_{\mathrm{h}})$ and $(1-\varepsilon)w(Q_{\mathrm{v}})$, respectively. Furthermore, the labels in Q'_{h} reach their positions in L'_{h} from their position in L by sliding horizontally without crossing a vertical stopping line. Thus, they do not interfere with the vertical movement that vertically sliding labels undergo in the transition from L to L'_{v}, and vice versa. Consequently, the union of L'_{h} and L'_{v} (defined in the obvious way) is a labeling of I of weight at least $(1 - \varepsilon)$ times the optimum. Applying the PTAS of Lemma 3.4 to $I_{\mathrm{h}} \cup I_{\mathrm{v}}$, we obtain a solution of I of weight at least $(1 - \varepsilon)w(Q'_{\mathrm{h}} \cup Q'_{\mathrm{v}}) \geq (1 - \varepsilon)^2 w(Q)$, which completes the proof. ∎

4. Open Problems

Corollary 2.2 states that a family of graphs is trimmable if it is of bounded treewidth *and* bounded degree. We cannot exclude, however, that the bounded-degree condition is superfluous. In other words, with $\mathbb{N} = \{1, 2, \ldots\}$, is there a function $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for all $k, t \in \mathbb{N}$, *every* weighted undirected graph of treewidth k has a $(t, g(k, t))$-trimming? The answer is yes in the unweighted case, i.e., if all weights are the same. If the answer were generally yes, it would follow by the argument in the proof of Corollary 2.3 that the family of planar graphs is also trimmable. More generally, the question of which families of graphs are trimmable deserves further study.

Acknowledgments

We thank Hans Bodlaender for pointing us to the concept of domino treewidth.

References

[AvKS98] Pankaj K. Agarwal, Marc van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Comput. Geom. Theory Appl.*, 11:209–218, 1998.

[Bak94] Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41:153–180, 1994.

[Bod98] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1–2):1–45, 1998.

[Bod99] Hans L. Bodlaender. A note on domino treewidth. *Discrete Math. Theor. Comput. Sci.*, 3(4):141–150, 1999.

[DO95] Guoli Ding and Bogdan Oporowski. Some results on tree decomposition of graphs. *J. Graph Theory*, 20:481–499, 1995.

[EHJ+06] Thomas Erlebach, Torben Hagerup, Klaus Jansen, Moritz Minzlaff, and Alexander Wolff. A new approximation algorithm for labeling weighted points with sliding labels. In *Proc. 22nd European Workshop on Computational Geometry (EWCG'06)*, pages 137–140, Delphi, 2006.

[PSS+03] Sheung-Hung Poon, Chan-Su Shin, Tycho Strijk, Takeaki Uno, and Alexander Wolff. Labeling points with weights. *Algorithmica*, 38(2):341–362, 2003.

[vKSW99] Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Comput. Geom. Theory Appl.*, 13:21–47, 1999.

[WS96] Alexander Wolff and Tycho Strijk. The Map-Labeling Bibliography. `http://i11www.ira.uka.de/map-labeling/bibliography`, 1996.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 277-288
www.stacs-conf.org

COMPUTING MINIMUM SPANNING TREES WITH UNCERTAINTY

THOMAS ERLEBACH [1], MICHAEL HOFFMANN [1], DANNY KRIZANC [2], MATÚŠ MIHAL'ÁK [3], AND RAJEEV RAMAN [1]

[1] Department of Computer Science, University of Leicester, UK.
E-mail address: {te17,mh55,rr29}@mcs.le.ac.uk

[2] Department of Mathematics and Computer Science, Wesleyan University, USA.
E-mail address: dkrizanc@wesleyan.edu

[3] Institut für Theoretische Informatik, ETH Zürich, Switzerland.
E-mail address: matus.mihalak@inf.ethz.ch

Abstract. We consider the minimum spanning tree problem in a setting where information about the edge weights of the given graph is uncertain. Initially, for each edge e of the graph only a set A_e, called an *uncertainty area*, that contains the actual edge weight w_e is known. The algorithm can 'update' e to obtain the edge weight $w_e \in A_e$. The task is to output the edge set of a minimum spanning tree after a minimum number of updates. An algorithm is k-update competitive if it makes at most k times as many updates as the optimum. We present a 2-update competitive algorithm if all areas A_e are open or trivial, which is the best possible among deterministic algorithms. The condition on the areas A_e is to exclude degenerate inputs for which no constant update competitive algorithm can exist.

Next, we consider a setting where the vertices of the graph correspond to points in Euclidean space and the weight of an edge is equal to the distance of its endpoints. The location of each point is initially given as an uncertainty area, and an update reveals the exact location of the point. We give a general relation between the edge uncertainty and the vertex uncertainty versions of a problem and use it to derive a 4-update competitive algorithm for the minimum spanning tree problem in the vertex uncertainty model. Again, we show that this is best possible among deterministic algorithms.

Key words and phrases: Algorithms and data structures; Current challenges: mobile and net computing.

Part of the presented research was undertaken while the second and fifth author were on study leave from the University of Leicester and the second author was visiting the University of Queensland. The authors would like to thank these institutions for their support.

1. Introduction

In many applications one has to deal with computational problems where some parts of the input data are imprecise or uncertain. For example, in a geometric problem involving sets of points, the locations of the points might be known only approximately; effectively this means that instead of the location of a point, only a region or area containing that point is known. In other applications, only estimates of certain input parameters may be known, for example in form of a probability distribution. There are many different approaches to dealing with problems of this type, including e.g. stochastic optimization and robust optimization.

Pursuing a different approach, we consider a setting in which the algorithm can obtain exact information about an input data item using an *update* operation, and we are interested in the *update complexity* of an algorithm, i.e., our goal is to compute a correct solution using a minimum number of updates. The updates are adaptive, i.e., one selects the next item to update based on the result of the updates performed so far, so we refer to the algorithm as an *on-line algorithm*. There are a number of application areas where this setting is meaningful. For example, in a mobile ad-hoc network an algorithm may have knowledge about the approximate locations of all nodes, and it is possible (but expensive) to find out the exact current location of a node by communicating to that node and requesting that information. To assess the performance of an algorithm, we compare the number of updates that the algorithm makes to the optimal number of updates. Here, optimality is defined in terms of an adversary, who, knowing the values of all input parameters, makes the fewest updates needed to present a solution to the problem that is provably correct, in that no additional areas need to be updated to verify the correctness of the solution claimed by the adversary. We say that an algorithm is *k-update competitive* if, for each input instance, the algorithm makes at most k times as many updates as the optimum number of updates for that input instance. The notions of update complexity and k-update competitive algorithms were implicit in Kahan's model for data in motion [6] and studied further for two-dimensional geometric problems by Bruce et al. [2].

In this paper, we consider the classical minimum spanning tree (MST) problem in two settings with uncertain information. In the first setting, the edge weights are initially given as uncertainty areas, and the algorithm can obtain the exact weight of an edge by updating the edge. If the uncertainty areas are trivial (i.e., contain a single number) or (topologically) open, we give a 2-update competitive algorithm and show that this is best possible for deterministic algorithms. Without this restriction on the areas, it is easy to construct degenerate inputs for which there is no constant update competitive algorithm. Although degeneracy could also be excluded by other means (similar to the "general position" assumption in computational geometry), our condition is much cleaner.

In the second setting that we consider, the vertices of the graph correspond to points in Euclidean space, and the locations of the points are initially given as uncertainty areas. The weight of an edge equals the distance between the points corresponding to its vertices. The algorithm can update a vertex to reveal its exact location. We give a general relation between the edge uncertainty version and the vertex uncertainty version of a problem. For trivial or open uncertainty areas we obtain a 4-update competitive algorithm for the MST problem with vertex uncertainty and show again that this is optimal for deterministic algorithms.

Related Work. We do not attempt to survey the vast literature dealing with problems on uncertain data, but focus on work most closely related to ours. Kahan [6] studied the problem of finding the maximum, the median and the minimum gap of a set of n real values constrained to fall in a given set of n real intervals. In the spirit of competitive analysis, he defined the *lucky ratio* of an update strategy as the worst-case ratio between the number of updates made by the strategy and the optimal number of updates of a non-deterministic strategy. In our terminology, a strategy with lucky ratio k is k-update competitive. Kahan gave strategies with optimal lucky ratios for the problems considered [6].

Bruce et al. studied the problems of computing maximal points or the points on the convex hull of a set of uncertain points [2] and presented 3-update competitive algorithms. They introduced a general method, called the *witness algorithm*, for dealing with problems involving uncertain data, and derived their 3-update competitive algorithms using that method. The algorithms we present in this paper are based on the method of the witness algorithm of [2], but the application to the MST problem is non-trivial.

Feder et al. [5, 4], consider two problems in a similar framework to ours. Firstly, they consider the problem of computing the median of n numbers to within a given tolerance. Each input number lies in an interval, and an update reveals the exact value, but different intervals have different update costs. They consider off-line algorithms, which must decide the sequence of updates prior to seeing the answers, as well as on-line ones, aiming to minimize the total update cost. In [4], off-line algorithms for computing the length of a shortest path from a source s to a given vertex t are considered. Again, the edge lengths lie in intervals with different update costs, and they study the computational complexity of minimizing the total update cost.

One difference between the framework of Feder et al. and ours is that they require the computation of a specific numeric value (the value of the median, the length of a shortest path). We, on the other hand, aim to obtain a subset of edges that form an MST. In general, our version of the problem may require far fewer updates. Indeed, for the MST with vertex uncertainties, it is obvious that one must update all non-trivial areas to compute the cost of the MST exactly. However, the cost of the MST may not be needed in many cases: if the MST is to be used as a routing structure in a wireless ad-hoc network, then it suffices to determine the edge set. Also, our algorithms aim towards on-line optimality against an adversary, whereas their off-line algorithms aim for static optimality.

Further work in this vein attempts to compute other aggregate functions to a given degree of tolerance, and establishes tradeoffs between update costs and error tolerance or presents complexity results for computing optimal strategies, see e.g. [10, 8].

Another line of work considers the robust spanning tree problem with interval data. For a given graph with weight intervals specified for its edges, the goal is to compute a spanning tree that minimizes the worst-case deviation from the minimum spanning tree (also called the *regret*), over all realizations of the edge weights. This is an off-line problem, and no update operations are involved. The problem is proved $\mathcal{NP}$-hard in [1]. A 2-approximation algorithm is given in [7]. Further work has considered heuristics or exact algorithms for the problem, see e.g. [12].

In the setting of geometric problems with imprecise points, Löffler and van Kreveld have studied the problem of computing the largest or smallest convex hull over all possible locations of the points inside their uncertainty areas [9]. Here, the option of updating a point does not exist, and the goal is to design fast algorithms computing an extremal solution over all possible choices of exact values of the input data.

The remainder of the paper is organized as follows. In Section 2, we define our problems and introduce the witness algorithm of [2] in general form. Sections 3 and 4 give our results for MSTs with edge and vertex uncertainty, respectively.

2. Preliminaries

The MST-EDGE-UNCERTAINTY problem is defined as follows: Let $G = (V, E)$ be a connected, undirected, weighted graph. Initially the edge weights w_e are unknown; instead, for each edge e an area A_e is given with $w_e \in A_e$. When updating an edge e, the value of w_e is revealed. The aim is to find (the edge set of) an MST for G with the least number of updates.

In applications such as mobile ad-hoc networks it is natural to assume the vertices of our graph are embedded in two or three dimensional space. This leads to the MST-VERTEX-UNCERTAINTY problem defined as follows: Let $G = (V, E)$ be a connected, undirected, weighted graph. The vertices correspond to points in Euclidean space. We refer to the point p_v corresponding to a vertex v as its *location*. The weight of an edge is the Euclidean distance between the locations of its vertices. Initially the locations of the vertices are not known; instead, for each vertex v an area A_v is given with $p_v \in A_v$, where p_v is the actual location of vertex v. When a vertex v is updated, the location p_v is revealed. The aim is to find an MST for G with the least number of updates.

Formally we are interested in on-line update problems of the following type: Each problem instance $P = (C, A, \phi)$ consists of an ordered set of data $C = \{c_1, \ldots, c_n\}$, also called a configuration, and a function ϕ such that $\phi(C)$ is the set of solutions for P. (The function ϕ is the same for all instances of a problem and can thus be taken to represent the problem.) At the beginning the set C is not known to the algorithm; instead, an ordered set of areas $A = \{A_1, \ldots, A_n\}$ is given, such that $c_i \in C$ is an element of A_i. The sets A_i are called *areas of uncertainty* or *uncertainty areas* for C. We say that an uncertainty area A_i that consists of a single element is *trivial*. For example, in the MST-EDGE-UNCERTAINTY problem, C consists of the given graph $G = (V, E)$ and its $|E|$ actual edge weights. The ordered set of areas A specifies the graph G exactly (so we assume complete knowledge of G) and, for each edge $e \in E$, contains an area A_e giving the possible values the weight of e may take. Then $\phi(C)$ is the set of MSTs of the graph with edge weights given by C, each tree represented as a set of edges.

For a given set of uncertainty areas $A = \{A_1, \ldots, A_n\}$, an area A_i can be *updated*, which reveals the exact value of c_i. After updating A_i, the new ordered set of areas of uncertainty for C is $\{A_1, \ldots, A_{i-1}, \{c_i\}, A_{i+1}, \ldots, A_n\}$. Updating all non-trivial areas would reveal the configuration C and would obviously allow us to calculate an element of $\phi(C)$ (under the natural assumption that ϕ is computable). The aim of the on-line algorithm is to minimize the number of updates needed in order to compute an element of $\phi(C)$.

An algorithm is k-update competitive for a given problem ϕ if for every problem instance $P = (C, A, \phi)$ the algorithm needs at most $k \cdot OPT + c$ updates, where c is a constant and OPT is the minimum number of updates needed to verify an element of $\phi(C)$. (For our algorithms we can take $c = 0$, but our lower bounds apply also to the case where c can be an arbitrary constant.) Note that the primary aim is to minimize the number of updates needed to calculate a solution. We do not consider running time or space requirements in detail, but note that our algorithms are clearly polynomial, provided that one can obtain

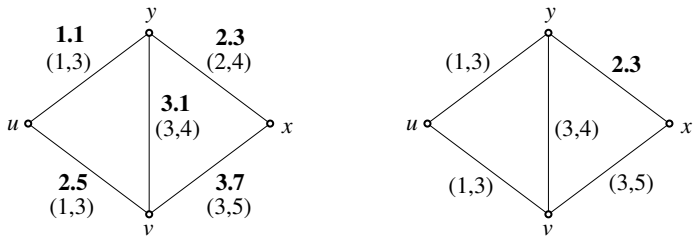


Figure 1: (a) Instance of MST-EDGE-UNCERTAINTY (b) Updating the edge $\{x, y\}$ suffices to verify an MST

if an element of $\phi(C)$ can not be calculated from A **then**
  find a witness set W
  update all areas in W
  let A' be the areas of uncertainty after updating W
  restart the algorithm with $P' = (C, A', \phi)$
end if
return an element of $\phi(C)$ that can be calculated from A

Figure 2: The general witness algorithm

the infimum and supremum of an area in $O(1)$ time, an assumption which holds e.g. if areas are open intervals.

As an example, consider the instance of MST-EDGE-UNCERTAINTY shown in Figure 1(a), where each edge is labeled with its actual weight (in bold) and its uncertainty area (an open interval). Updating the edge $\{x, y\}$ leads to the situation shown in Figure 1(b) and suffices to verify that the edges $\{u, y\}$, $\{u, v\}$ and $\{x, y\}$ form an MST regardless of the exact weights of the edges that have not yet been updated. If no edge is updated, one cannot exclude that an MST includes the edge $\{v, x\}$ instead of $\{x, y\}$, as the former could have weight 3.3 and the latter weight 3.9, for example. Therefore, for the instance of MST-EDGE-UNCERTAINTY in Figure 1(a) the minimum number of updates is 1.

2.1. The Witness Algorithm

The witness algorithm for problems with uncertain input was first introduced in [2]. This section describes the witness algorithm in a more general setting and notes some of its properties. We call $W \subseteq A$ a *witness set* of (A, ϕ) if for every possible configuration C (where $c_i \in A_i$) no element of $\phi(C)$ can be verified without updating an element of W. In other words, any set of updates that suffices to verify a solution must update at least one area of W. The witness algorithm for a problem instance $P = (C, A, \phi)$ is shown in Figure 2.

For two ordered sets of areas $A = \{A_1, A_2, \ldots, A_n\}$ and $B = \{B_1, B_2, \ldots, B_n\}$ we say that B is at least as narrow as A if $B_i \subseteq A_i$ for all $1 \leq i \leq n$. The following lemma is easy to prove.

Lemma 2.1. *Let $P = (C, A, \phi)$ be a problem instance and B be a narrower set of areas than A. Further let W be a witness set of (B, ϕ). Then W is also a witness set of (A, ϕ).*

Theorem 2.2. *If there is a global bound k on the size of any witness set used by the witness algorithm, then the witness algorithm is k-update competitive.*

Theorem 2.2 was proved in a slightly different setting in [2], but the proof carries over to the present setting in a straightforward way by using Lemma 2.1.

3. Minimum Spanning Trees with Edge Uncertainty

In this section we present an algorithm U-RED for the problem MST-EDGE-UNCERTAINTY. In the case that all areas of uncertainty are either open or trivial, algorithm U-RED is 2-update competitive, which we show is optimal. Furthermore, we show that for arbitrary areas of uncertainty there is no constant update competitive algorithm.

First, let us recall a well known property, usually referred to as the *red rule* [11], of MSTs:

Proposition 3.1. *Let G be a weighted graph and let C be a cycle in G. If there exists an edge $e \in C$ with $w_e > w_{e'}$ for all $e' \in C - \{e\}$, then e is not in any MST of G.*

We will use the following notations and definitions: A graph $\mathcal{U} = (V, E)$ with an area A_e for each edge $e \in E$ is called an *edge-uncertainty graph*. We say a weighted graph $G = (V, E)$ with edge weights w_e is a *realization* of $\mathcal{U}$ if $w_e \in A_e$ for every $e \in E$. Note that w_e is associated with G and A_e with $\mathcal{U}$. We also say that an edge e is *trivial* if the area A_e is trivial.

For an edge e in an edge-uncertainty graph we denote the upper limit of A_e by $U_e = \sup A_e$ and the lower limit of A_e by $L_e = \inf A_e$.

We extend the notion of an MST to edge-uncertainty graphs in the following way: Let $\mathcal{U}$ be an edge-uncertainty graph. We say T is an *MST of* $\mathcal{U}$ if T is an MST of every realization of $\mathcal{U}$. Clearly not every edge-uncertainty graph has an MST.

Let C be a cycle in $\mathcal{U}$. We say the edge $e \in C$ is an *always maximal* edge in C if $L_e \geq U_c$ for all $c \in C - \{e\}$. Therefore in every realization G of $\mathcal{U}$ we have $w_e \geq w_c$ for all $c \in C - \{e\}$.

Note that a cycle can have more than one always maximal edge and not every cycle has an always maximal edge. The following lemma deals with cycles of the latter kind:

Lemma 3.2. *Let $\mathcal{U}$ be an edge-uncertainty graph. Let C be a cycle in $\mathcal{U}$. Let C not have an always maximal edge. Then for any $f \in C$ with $U_f = \max\{U_c \mid c \in C\}$ we have that f is non-trivial and there exists an edge $g \in C - \{f\}$ with $U_g > L_f$.*

Proof. Let $f \in C$ be an edge with $U_f = \max\{U_c \mid c \in C\}$. If $L_f = U_f$ the edge f would be always maximal. Hence L_f must be strictly smaller than U_f and f is non-trivial. Since there is no always maximal edge in C, we have that $L_f < \max\{U_c \mid c \in C - \{f\}\}$. Therefore there exists at least one edge g in $C - \{f\}$ with $L_f < U_g$. ∎

Proposition 3.3. *Let $\mathcal{U}$ be an edge-uncertainty graph with an MST T. Let $f = \{u, v\}$ be an edge of $\mathcal{U}$ such that $f \notin T$. Let P be the path in T connecting u and v, then $U_p \leq L_f$ for all $p \in P$.*

Proof. Assume there exists a $p \in P$ with $U_p > L_f$. Then there exists a realization G of $\mathcal{U}$ with $w_p > w_f$. Hence by removing the edge p and adding the edge f to T we obtain a spanning tree that is cheaper than T. So T is not an MST for G. This is a contradiction since T is an MST of $\mathcal{U}$ and therefore of any realization of $\mathcal{U}$. ∎

```
01 Index all edges such that e_1 ≤ e_2 ≤ ··· ≤ e_m.
02 Let Γ be U without any edge
03 for i from 1 to m do
04         add e_i to Γ
05         if Γ has a cycle C then
06                 if C contains an always maximal edge e then
07                         delete e from Γ
08                 else
09                         let f ∈ C such that U_f = max{U_c|c ∈ C}
10                         let g ∈ C − {f} such that U_g > L_f
11                         update f and g
12                         restart the algorithm
13                 end if
14         end if
15 end for
16 return Γ
```

Figure 3: Algorithm U-RED

Our algorithm U-RED applies the red rule to the given uncertainty graph, but we have to be careful about the order in which edges are considered. The order we use is as follows: Let $\mathcal{U}$ be an edge-uncertainty graph and let e, f be two edges in $\mathcal{U}$. We say

$e < f$ if $L_e < L_f$ or ($L_e = L_f$ and $U_e < U_f$),
$e \leq f$ if $e < f$ or ($L_e = L_f$ and $U_e = U_f$).

Edges with the same upper and lower weight limit are ordered arbitrarily.

Algorithm U-RED is shown in Figure 3. Observe that:

- In case no update is made the algorithm U-RED will perform essentially Kruskal's algorithm [3]. When a cycle is created there will be an always maximal edge in that cycle. Due to the order in which the algorithm adds the edges to Γ the edge e_i that closes a cycle C must be an always maximal edge in C. So where Kruskal's algorithm does not add an edge to Γ when it would close a cycle, the U-RED algorithm adds this edge to Γ but then deletes it or an equally weighted edge in the cycle from Γ.
- By Lemma 3.2 the edges f, g in line 9 and 10 exist and f is non-trivial.
- The algorithm will terminate. The algorithm either updates at least one non-trivial edge f and restarts, or does not perform any updates. Hence the algorithm U-RED will eventually return an MST of G.
- During the run of the algorithm the graph Γ is either a forest or contains one cycle. In case the most recently added edge closes a cycle either one edge of the cycle will be deleted or after some updates the algorithm restarts and Γ has no edges. Hence at any given time there is at most one cycle in Γ.

As the algorithm may restart itself, we say a *run* is completed if the algorithm restarts or returns the MST. In case of a restart, another run of the algorithm starts.

Before showing that the algorithm U-RED is 2-update competitive under the restriction to open or trivial areas, we discuss some technical preliminaries. In each run the algorithm considers all edges in a certain order $e_1, \ldots, e_m$. During the run of the algorithm we refer to the currently considered edge as e_i. Let u and v be two distinct vertices. In case u and v are in the same connected component of the subgraph with edges $e_1, \ldots, e_{i-1}$, then

they are also connected in the current Γ. Furthermore, we need some properties of a path connecting u and v in Γ under certain conditions. The next two lemmas establish these properties. They are technical and are solely needed in the proof of Lemma 3.6.

Lemma 3.4. *Let $h = \{u, v\}$ and e be two edges in $\mathcal{U}$. Let $h \neq e$ and $L_h < U_e$. Let the algorithm be in a state such that h has been considered. Then u and v are connected in the current $\Gamma - \{e\}$.*

Proof. If the edge h is in the current Γ then clearly u and v are connected in $\Gamma - \{e\}$, so assume that h is no longer in Γ. Therefore it must have been an always maximal edge in a cycle C. In order for h to be an always maximal edge in C we must have that $L_c \leq U_c \leq L_h$ for all $c \in C - \{h\}$. So since $L_h < U_e$ we have that $L_c < U_e$. Also the edge h can not be an always maximal edge in C if C contains e.

Clearly $C - \{h\}$ is a path in U connecting u and v and does not contain e. Since the edges in $C - \{h\}$ might have been deleted from the current Γ themselves we have to use this argument repeatedly, but eventually we get a path in the current $\Gamma - \{e\}$ connecting u and v. ∎

The next lemma follows directly from Lemma 3.4.

Lemma 3.5. *Let u, v be vertices and e be an edge in $\mathcal{U}$. Let P be a path in $\mathcal{U} - \{e\}$ connecting u and v with $L_p < U_e$ for all $p \in P$. Let the algorithm be in a state such that all edges of P have been considered, then there exists a path P' in the current Γ connecting u and v with $e \notin P'$.*

Lemma 3.6. *Assume that all uncertainty areas are open or trivial. The edges f and g as described in the algorithm* U-RED *at line* 9 *and* 10 *form a witness set.*

Proof. We have the following situation: There exist a cycle C in Γ with no always maximal edge. Let $m = \max\{U_c \mid c \in C\}$. The edges f and g are in C with $U_f = m$ and $U_g > L_f$. By Lemma 3.2 the area A_f is non-trivial.

We now assume that the set $\{f, g\}$ is not a witness set. So we can update some edges, but not f or g such that the resulting edge-uncertainty graph $\mathcal{U}'$ has an MST T. Let U'_e and L'_e denote the upper and lower limit of an area for an edge e with regard to $\mathcal{U}'$. Since both edges f and g are not updated we note that

$$L_f = L'_f, U_f = U'_f, L_g = L'_g, U_g = U'_g.$$

Since all areas in $\mathcal{U}'$ and $\mathcal{U}$ are either trivial or open, and C has no always maximal edge, the weight of every edge in C must be less than m. In particular we have that for all $c \in C$

$$U'_c < m \text{ or } L'_c < U_c = m.$$

Since $U_f = m$ there exists a realization G' of $\mathcal{U}'$ and $\mathcal{U}$, where the weight of f is greater than the weight of any other edge in C. By Proposition 3.1 the edge f is not in any MST of G' and therefore also not in T.

Let u and v be the vertices of f. By Proposition 3.3 there exists a path P in $\mathcal{U}'$ connecting u and v with $U'_p \leq L_f$ for all $p \in P$. Since $U_g > L_f$ and neither f nor g are updated the edge g is not in the path P. We now argue that all edges of P must have been already considered by the algorithm. For this we look at the following two cases:
Case 1) Let $p \in P$ and $L'_p < L_f$. Since $L_p \leq L'_p$ we have that $L_p < L_f$.

Case 2) Let $p \in P$ and $L'_p = L_f$. Since $U'_p \leq L_f$ we have that $L'_p = U'_p = L_f$. Either the area A_p is also trivial ($L_p = U_p = L'_p = U'_p = L_f$) or A_p is open and contains the point L'_p, in this case $L_p < L'_p$.

So for all $p \in P$ we have $L_p < L_f$ or $L_p = U_p = L_f < U_f$. Therefore all edges of P will be considered before f. We also note that $L_p \leq L'_p \leq L_f < U_g$ for all $p \in P$. By Lemma 3.5 there exists a path P' in Γ connecting u and v and $g \notin P'$. Hence Γ has two cycles, which is a contradiction. ∎

Using Theorem 2.2, this leads directly to the following result.

Theorem 3.7. *Under the restriction to open and trivial areas the algorithm* U-RED *is* 2*-update competitive.*

We remark that the analysis of algorithm U-RED actually works also in the more general setting where it is only required that every area is trivial or satisfies the following condition: the area contains neither its infimum nor its supremum. It remains to show that under the restriction to open and trivial areas there is no algorithm for the MST-EDGE-UNCERTAINTY problem that is $(2-\epsilon)$-update competitive.

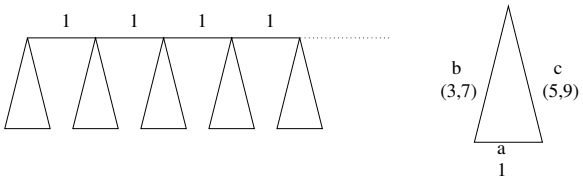


Figure 4: Lower bound construction

Example 3.8. The graph G displayed in Figure 4 consists of a path and, for each vertex of the path, a gadget connected to that vertex. Each gadget is a triangle with sides a, b and c and areas $A_a = \{1\}$, $A_b = (3, 7)$ and $A_c = (5, 9)$. In each gadget a and either b or c are part of the minimum spanning tree. If the algorithm updates b we let the weight of b be 6. So c needs to be updated, which reveals a weight for c of 8. However, by updating only c the edge b would be part of the minimum spanning tree regardless of its exact weight. If the algorithm updates c first, we let the weight of c be 6. The necessary update of b reveals a weight of 4, and updating only b would have been enough. So in each gadget every algorithm makes two updates where only one is needed by OPT. Hence no deterministic algorithm is $(2-\epsilon)$-update competitive.

The following example shows that without restrictions on the areas there is no algorithm for the MST-EDGE-UNCERTAINTY problem that is constant update competitive.

Example 3.9. Figure 5(a) shows an example of an edge-uncertainty graph for which no algorithm can be constant update competitive. The minimum spanning tree consists of all edges incident with u and all edges incident with v plus one more edge. Let us assume the weight of one of the remaining $k = (n-2)/2$ edges is 2 and the weight of the others is 3. Any algorithm would need to update these edges until it finds the edge with weight 2. This in the worst case could be the last edge and k updates were made. However OPT will only update the edge with weight 2 and therefore $OPT = 1$.

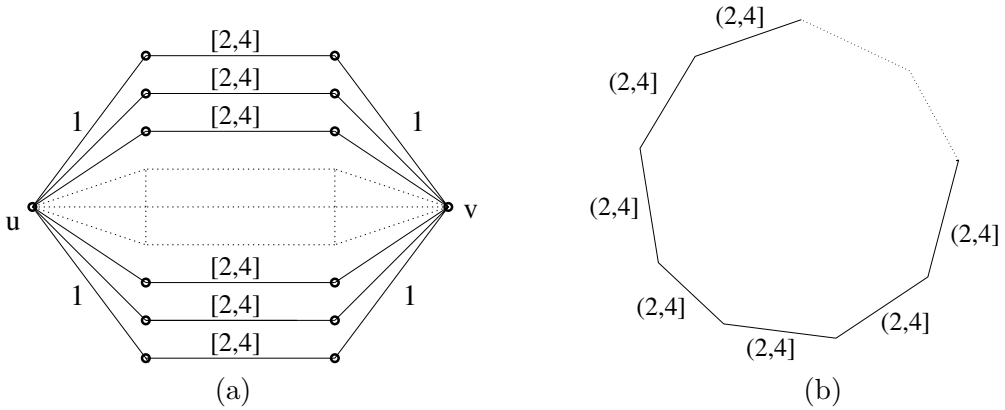


Figure 5: Non-existence of constant update competitive algorithms

Note that this example actually shows that there is no algorithm that is better then $(n-2)/2$-update competitive, where n is the number of vertices of the given graph. By adding edges with uncertainty area $[2,4]$ such that the neighbors of u and the neighbors of v form a complete bipartite graph, we even get a lower bound of $\Omega(n^2)$.

The construction in Example 3.9 works also if the intervals $[2,4]$ are replaced by half-open intervals $[2,4)$. Thus, the example demonstrates that with closed lower limits on the areas there is no constant update competitive algorithm for the MST-EDGE-UNCERTAINTY problem. The following example does the same for closed upper limits.

Example 3.10. The graph shown in Figure 5(b) is one big cycle with k edges and the uncertainty area of each edge is $(2,4]$. Let us assume exactly one edge e has weight 4 and the others are of weight 3. In the worst case any algorithm has to update all k edges before finding e. However OPT is 1 by just updating e.

4. Minimum Spanning Tree with Vertex Uncertainty

In this section we consider the model of vertex-uncertainty graphs. The models of vertex-uncertainty and edge-uncertainty are closely related. Clearly a vertex uncertainty graph $\mathcal{U}$ has an associated edge-uncertainty graph $\bar{\mathcal{U}}$ where the area for each edge $e=\{u,v\}$ is determined by the combinations of possible locations of u and v in $\mathcal{U}$, i.e., the areas $\bar{A}$ in $\bar{\mathcal{U}}$ are defined as $\bar{A}_{\{u,v\}} = \{d(u',v') | u' \in A_u, v' \in A_v\}$.

An update of an edge $e=\{u,v\}$ in $\bar{\mathcal{U}}$ can be performed (simulated) by updating u and v in $\mathcal{U}$; these two vertex updates might also reveal additional information about the weights of other edges incident with u or v. Furthermore, note that if neither of the two vertices u and v in $\mathcal{U}$ is updated, no information about the weight of e can be obtained. Thus, we get:

Lemma 4.1. *Let ϕ be a graph problem such that the set of solutions for a given edge-weighted graph $G=(V,E)$ depends only on the graph and the edge weights (but not the locations of the vertices). Let $\mathcal{U}$ be a vertex-uncertainty graph that is an instance of ϕ. If $\bar{W} \subseteq E$ is a witness set for $\bar{\mathcal{U}}$, then $W = \bigcup_{\{u,v\}\in\bar{W}}\{u,v\}$ is a witness set for $\mathcal{U}$.*

Using Theorem 2.2 we obtain the following result.

Theorem 4.2. *Let ϕ be a graph problem such that the set of solutions for a given edge-weighted graph depends only on the graph and the edge weights (but not the locations of*

the vertices). Let A be a k-update competitive algorithm for the problem ϕ with respect to edge-uncertainty graphs. If A is a witness algorithm, then by simulating an edge update by updating both its endpoints the algorithm A is $2k$-update competitive for vertex-uncertainty graphs.

By standard properties of Euclidean topology, the following lemma clearly holds.

Lemma 4.3. *Let $\mathcal{U}$ be a vertex uncertainty graph with only trivial or open areas. Then $\bar{\mathcal{U}}$ also has only trivial or open areas.*

Theorem 4.4. *Under the restriction to trivial or open areas the algorithm* U-RED *is* 4*-update competitive for the* MST-VERTEX-UNCERTAINTY *problem, which is optimal.*

Proof. Combining Theorems 3.7 and 4.2 together with Lemma 4.3, we get that U-RED is 4-update competitive for the MST-VERTEX-UNCERTAINTY problem when restricted to trivial or open areas. It remains to show that this is optimal.

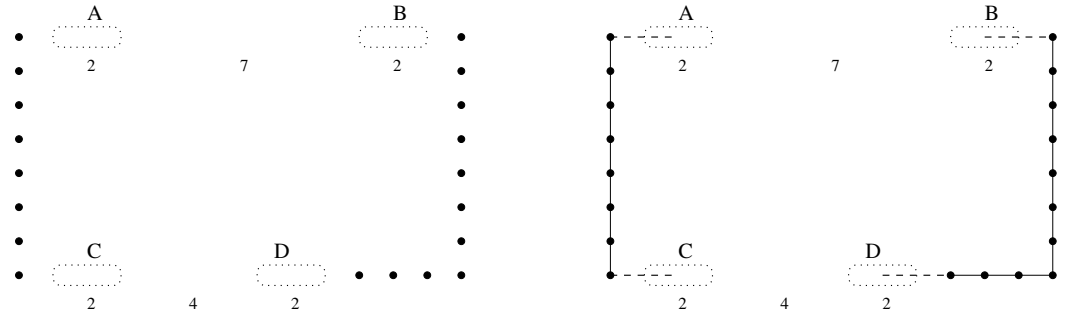


Figure 6: (a) Lower bound construction (b) Edges that are in any minimum spanning tree

We show that no algorithm can be better than 4-update competitive. In Figure 6(a) we give a construction in the Euclidean plane for which any algorithm can be forced to make 4 updates, while OPT is 1. The black dots on the left and right represent trivial areas. The distance between two neighboring trivial areas is 1. There are four non-trivial areas A, B, C and D. Each of these areas is a long, thin open area of length 2 and small positive width. The distance between each non-trivial area and its closest trivial area is 1 as well. Let G be the complete graph with one vertex for each of the trivial and non-trivial areas.

Independent of the exact locations of the vertices in the non-trivial areas A, B, C and D, the edges indicated in Figure 6(b) must be part of any MST. Note that the distance between the vertex of a non-trivial area and its trivial neighbor is in $(1, 3)$ and thus less than 3.

We now consider the distances between the non-trivial areas. We let $d(X, Y)$ be the area of all possible distances between two vertex areas X and Y. So $d(A, B) = (7, 11)$, $d(C, D) = (4, 8)$. Note that the distance between the vertices in A and D and the distance between the vertices in B and C are greater than 8, so either the edge AB or the edge CD is part of the minimum spanning tree.

Every algorithm will update the areas A, B, C and D in a certain order until it is clear that either the distance between the vertices of A and B is smaller or equal to the distance between the vertices of C and D, or vice versa. In order to force the algorithm to update all four areas, we let the locations of the vertices revealed in any of the first 3 updates made by the algorithm be as follows:

- A or D: the vertex will be located far to the right,
- B or C: the vertex will be located far to the left.

Here, 'far to the right' or 'far to the left' means that the location is very close (distance $\varepsilon > 0$, for some small ε) to the right or left end of the area, respectively.

We show that it is impossible for the algorithm to output a correct minimum spanning tree after only three updates. Consider the situation after the algorithm has updated three of the four non-trivial areas. Since the choice of the locations of the vertices in the areas is independent of the sequence of updates, we have to consider four cases depending on which of the four areas has not yet been updated. We use A', B', C' and D' to refer to the areas A, B, C and D after they have been updated. If the area A is the only area that has not yet been updated, we have that $d(A, B') = (7 + \epsilon, 9 + \epsilon)$ and $d(C', D') = \{8 - 2\epsilon\}$. Clearly the area A needs to be updated. By having the vertex of area A on the far left, updating only area A instead of the areas B, C, D results in $d(A', B) = (9 - \epsilon, 11 - \epsilon)$ and $d(C, D) = (4, 8)$. Hence OPT would only update the area A and know that the edge AB is not part of the minimum spanning tree. The other three cases are similar. So for the construction in Figure 6(a), no algorithm can guarantee to make less than 4 updates even though a single update is enough for the optimum. Furthermore, we can create k disjoint copies of the construction and connect them using lines of trivial areas spaced 1 apart. As long as the copies are sufficiently far apart, they will not interfere with each other. Hence, for a graph with k copies there is no algorithm that can guarantee less than $4k$ updates when at the same time $OPT = k$. ∎

References

[1] I. Aron and P. Van Hentenryck. On the complexity of the robust spanning tree problem with interval data. *Operations Research Letters*, 32(1):36–40, 2004.

[2] R. Bruce, M. Hoffmann, D. Krizanc, and R. Raman. Efficient update strategies for geometric computing with uncertainty. *Theory of Computing Systems*, 38(4):411–423, 2005.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[4] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy. Computing shortest paths with uncertainty. *Journal of Algorithms*, 62(1):1–18, 2007.

[5] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. *SIAM Journal on Computing*, 32(2):538–547, 2003.

[6] S. Kahan. A model for data in motion. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC'91)*, pp. 267–277, 1991.

[7] A. Kasperski and P. Zieliński. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Information Processing Letters*, 97(5):177–180, 2006.

[8] S. Khanna and W.-C. Tan. On computing functions with uncertainty. In *Proceedings of the 20th Symposium on Principles of Database Systems (PODS'01)*, pp. 171–182, 2001.

[9] M. Löffler and M. J. van Kreveld. Largest and smallest tours and convex hulls for imprecise points. In *10th Scand. Workshop on Algorithm Theory (SWAT'06)*, LNCS 4059, pp. 375–387. Springer, 2006.

[10] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. In *Proc. 26th Intern. Conference on Very Large Data Bases (VLDB'00)*, pp. 144–155, 2000.

[11] R. E. Tarjan. *Data structures and network algorithms*. SIAM, Philadelphia, PA, 1983.

[12] H. Yaman, O. Karasan, and M. Pinar. The robust spanning tree problem with interval data. *Operations Research Letters*, 29(1):31–40, 2001.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 289-300
www.stacs-conf.org

CONVERGENCE THRESHOLDS OF NEWTON'S METHOD FOR MONOTONE POLYNOMIAL EQUATIONS

JAVIER ESPARZA, STEFAN KIEFER, AND MICHAEL LUTTENBERGER

Institut für Informatik
Technische Universität München, Germany
E-mail address: {esparza,kiefer,luttenbe}@in.tum.de

Abstract. Monotone systems of polynomial equations (MSPEs) are systems of fixed-point equations $X_1 = f_1(X_1, \ldots, X_n), \ldots, X_n = f_n(X_1, \ldots, X_n)$ where each f_i is a polynomial with positive real coefficients. The question of computing the least non-negative solution of a given MSPE $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ arises naturally in the analysis of stochastic models such as stochastic context-free grammars, probabilistic pushdown automata, and back-button processes. Etessami and Yannakakis have recently adapted Newton's iterative method to MSPEs. In a previous paper we have proved the existence of a threshold $k_{\boldsymbol{f}}$ for strongly connected MSPEs, such that after $k_{\boldsymbol{f}}$ iterations of Newton's method each new iteration computes at least 1 new bit of the solution. However, the proof was purely existential. In this paper we give an upper bound for $k_{\boldsymbol{f}}$ as a function of the minimal component of the least fixed-point $\mu\boldsymbol{f}$ of $\boldsymbol{f}(\boldsymbol{X})$. Using this result we show that $k_{\boldsymbol{f}}$ is at most single exponential resp. linear for strongly connected MSPEs derived from probabilistic pushdown automata resp. from back-button processes. Further, we prove the existence of a threshold for arbitrary MSPEs after which each new iteration computes at least $1/w2^h$ new bits of the solution, where w and h are the width and height of the DAG of strongly connected components.

1. Introduction

A *monotone system of polynomial equations* (MSPE for short) has the form

$$\begin{aligned} X_1 &= f_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= f_n(X_1, \ldots, X_n) \end{aligned}$$

where $f_1, \ldots, f_n$ are polynomials with *positive* real coefficients. In vector form we denote an MSPE by $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$. We call MSPEs "monotone" because $\boldsymbol{x} \leq \boldsymbol{x'}$ implies $\boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{f}(\boldsymbol{x'})$

1998 ACM Subject Classification: G.1.5, Mathematics of Computing, Numerical Analysis.
Key words and phrases: Newton's Method, Fixed-Point Equations, Formal Verification of Software, Probabilistic Pushdown Systems.
This work was supported by the project *Algorithms for Software Model Checking* of the *Deutsche Forschungsgemeinschaft (DFG)*. Part of this work was done at the Universität Stuttgart.

for every $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^n_{\geq 0}$. MSPEs appear naturally in the analysis of many stochastic models, such as context-free grammars (with numerous applications to natural language processing [19, 15], and computational biology [21, 4, 3, 17]), probabilistic programs with procedures [6, 2, 10, 8, 7, 9, 11], and web-surfing models with back buttons [13, 14].

By Kleene's theorem, a feasible MSPE $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ (i.e., an MSPE with at least one solution) has a least solution $\mu\boldsymbol{f}$; this solution can be irrational and non-expressible by radicals. Given an MSPE and a vector $\boldsymbol{v}$ encoded in binary, the problem whether $\mu\boldsymbol{f} \leq \boldsymbol{v}$ holds is in PSPACE and at least as hard as the SQUARE-ROOT-SUM problem, a well-known problem of computational geometry (see [10, 12] for more details).

For the applications mentioned above the most important question is the efficient numerical approximation of the least solution. Finding the least solution of a feasible system $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ amounts to finding the least solution of $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{0}$ for $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{f}(\boldsymbol{X}) - \boldsymbol{X}$. For this we can apply (the multivariate version of) *Newton's method* [20]: starting at some $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$ (we use uppercase to denote variables and lowercase to denote values), compute the sequence

$$\boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} - (\boldsymbol{F}'(\boldsymbol{x}^{(k)}))^{-1}\boldsymbol{F}(\boldsymbol{x}^{(k)})$$

where $\boldsymbol{F}'(\boldsymbol{X})$ is the Jacobian matrix of partial derivatives.

While in general the method may not even be defined ($\boldsymbol{F}'(\boldsymbol{x}^{(k)})$ may be singular for some k), Etessami and Yannakakis proved in [10, 12] that this is not the case for the *Decomposed Newton's Method (DNM)*, that decomposes the MSPE into *strongly connected components* (SCCs) and applies Newton's method to them in a bottom-up fashion[1].

The results of [10, 12] provide no information on the number of iterations needed to compute i *valid bits* of $\mu\boldsymbol{f}$, i.e., to compute a vector $\boldsymbol{\nu}$ such that $\left|\mu\boldsymbol{f}_j - \nu_j\right| / \left|\mu\boldsymbol{f}_j\right| \leq 2^{-i}$ for every $1 \leq j \leq n$. In a former paper [16] we have obtained a first positive result on this problem. We have proved that for every strongly connected MSPE $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ there exists a threshold $k_{\boldsymbol{f}}$ such that for every $i \geq 0$ the $(k_{\boldsymbol{f}} + i)$-th iteration of Newton's method has at least i valid bits of $\mu\boldsymbol{f}$. So, loosely speaking, after $k_{\boldsymbol{f}}$ iterations DNM is guaranteed to compute at least 1 new bit of the solution per iteration; we say that DNM converges *linearly with rate 1.*

The problem with this result is that its proof provides no information on $k_{\boldsymbol{f}}$ other than its existence. In this paper we show that the threshold $k_{\boldsymbol{f}}$ can be chosen as

$$k_{\boldsymbol{f}} = 3n^2m + 2n^2\left|\log \mu_{\min}\right|$$

where n is the number of equations of the MSPE, m is such that all coefficients of the MSPE can be given as ratios of m-bit integers, and $\mu_{\min}$ is the minimal component of the least solution $\mu\boldsymbol{f}$.

It can be objected that $k_{\boldsymbol{f}}$ depends on $\mu\boldsymbol{f}$, which is precisely what Newton's method should compute. However, for MSPEs coming from stochastic models, such as the ones listed above, we can do far better. The following observations and results help to deal with $\mu_{\min}$:

- We obtain a syntactic bound on $\mu_{\min}$ for probabilistic programs with procedures (having stochastic context-free grammars and back-button stochastic processes as special instances) and prove that in this case $k_{\boldsymbol{f}} \leq n2^{n+2}m$.

[1] A subset of variables and their associated equations form an SCC, if the value of any variable in the subset influences the value of all variables in the subset, see Section 2 for details.

- We show that if every procedure has a non-zero probability of terminating, then $k_{\boldsymbol{f}} \leq 3nm$. This condition always holds in the special case of back-button processes [13, 14]. Hence, our result shows that i valid bits can be computed in time $\mathcal{O}((nm+i)\cdot n^3)$ in the unit cost model of Blum, Shub and Smale [1], where each single arithmetic operation over the reals can be carried out exactly and in constant time. It was proved in [13, 14] by a reduction to a semidefinite programming problem that i valid bits can be computed in poly(i,n,m)-time in the classical (Turing-machine based) computation model. We do not improve this result, because we do not have a proof that round-off errors (which are inevitable on Turing-machine based models) do not crucially affect the convergence of Newton's method. But our result sheds light on the convergence of a practical method to compute $\mu\boldsymbol{f}$.
- Finally, since $\boldsymbol{x}^{(k)} \leq \boldsymbol{x}^{(k+1)} \leq \mu\boldsymbol{f}$ holds for every $k \geq 0$, as Newton's method proceeds it provides better and better lower bounds for $\mu_{\min}$ and thus for $k_{\boldsymbol{f}}$. In the paper we exhibit a MSPE for which, using this fact and our theorem, we can prove that no component of the solution reaches the value 1. This cannot be proved by just computing more iterations, no matter how many.

The paper contains two further results concerning non-strongly-connected MSPEs: Firstly, we show that DNM still converges linearly even if the MSPE has more than one SCC, albeit the convergence rate is poorer. Secondly, we prove that Newton's method is well-defined also for non-strongly-connected MSPEs. Thus, it is not necessary to decompose an MSPE into its SCCs – although decomposing the MSPE may be preferred for efficiency reasons.

The paper is structured as follows. In Section 2 we state preliminaries and give some background on Newton's method applied to MSPEs. Sections 3, 5, and 6 contain the three results of the paper. Section 4 contains applications of our main result. We conclude in Section 7. Missing proofs can be found in a technical report [5].

2. Preliminaries

In this section we introduce our notation and formalize the concepts mentioned in the introduction.

2.1. Notation

$\mathbb{R}$ and $\mathbb{N}$ denote the sets of real, respectively natural numbers. We assume $0 \in \mathbb{N}$. $\mathbb{R}^n$ denotes the set of n-dimensional real valued *column* vectors and $\mathbb{R}^n_{\geq 0}$ the subset of vectors with non-negative components. We use bold letters for vectors, e.g. $\boldsymbol{x} \in \mathbb{R}^n$, where we assume that $\boldsymbol{x}$ has the components $x_1, \ldots, x_n$. Similarly, the i^{th} component of a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ is denoted by f_i.

$\mathbb{R}^{m\times n}$ denotes the set of matrices having m rows and n columns. The transpose of a vector or matrix is indicated by the superscript $^\top$. The identity matrix of $\mathbb{R}^{n\times n}$ is denoted by Id.

The *formal Neumann series* of $A \in \mathbb{R}^{n\times n}$ is defined by $A^* = \sum_{k\in\mathbb{N}} A^k$. It is well-known that A^* exists if and only if the spectral radius of A is less than 1, i.e. $\max\{|\lambda| \mid \mathbb{C} \ni \lambda \text{ is an eigenvalue of } A\} < 1$. If A^* exists, we have $A^* = (\mathrm{Id} - A)^{-1}$.

The partial order $\leq$ on $\mathbb{R}^n$ is defined as usual by setting $\boldsymbol{x} \leq \boldsymbol{y}$ if $x_i \leq y_i$ for all $1 \leq i \leq n$. By $\boldsymbol{x} < \boldsymbol{y}$ we mean $\boldsymbol{x} \leq \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$. Finally, we write $\boldsymbol{x} \prec \boldsymbol{y}$ if $x_i < y_i$ in every component.

We use $X_1, \ldots, X_n$ as variable identifiers and arrange them into the vector $\boldsymbol{X}$. In the following n always denotes the number of variables, i.e. the dimension of $\boldsymbol{X}$. While $\boldsymbol{x}, \boldsymbol{y}, \ldots$ denote arbitrary elements in $\mathbb{R}^n$, resp. $\mathbb{R}^n_{\geq 0}$, we write $\boldsymbol{X}$ if we want to emphasize that a function is given w.r.t. these variables. Hence, $\boldsymbol{f}(\boldsymbol{X})$ represents the function itself, whereas $\boldsymbol{f}(\boldsymbol{x})$ denotes its value for $\boldsymbol{x} \in \mathbb{R}^n$.

If Y is a set of variables and $\boldsymbol{x}$ a vector, then by $\boldsymbol{x}_Y$ we mean the vector obtained by restricting $\boldsymbol{x}$ to the components in Y.

The *Jacobian* of a differentiable function $\boldsymbol{f}(\boldsymbol{X})$ with $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is the matrix $\boldsymbol{f}'(\boldsymbol{X})$ given by

$$\boldsymbol{f}'(\boldsymbol{X}) = \begin{pmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_1}{\partial X_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial X_1} & \cdots & \frac{\partial f_m}{\partial X_n} \end{pmatrix} .$$

2.2. Monotone Systems of Polynomials

Definition 2.1. A function $\boldsymbol{f}(\boldsymbol{X})$ with $\boldsymbol{f} : \mathbb{R}^n_{\geq 0} \to \mathbb{R}^n_{\geq 0}$ is a *monotone system of polynomials (MSP)*, if every component $f_i(\boldsymbol{X})$ is a polynomial in the variables $X_1, \ldots, X_n$ with coefficients in $\mathbb{R}_{\geq 0}$. We call an MSP $\boldsymbol{f}(\boldsymbol{X})$ *feasible* if $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{y})$ for some $\boldsymbol{y} \in \mathbb{R}^n_{\geq 0}$.

Fact 2.2. *Every MSP $\boldsymbol{f}$ is monotone on $\mathbb{R}^n_{\geq 0}$, i.e. for $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{y}$ we have $\boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{f}(\boldsymbol{y})$.*

Since every MSP is continuous, Kleene's fixed-point theorem (see e.g. [18]) applies.

Theorem 2.3 (Kleene's fixed-point theorem)**.** *Every feasible MSP $\boldsymbol{f}(\boldsymbol{X})$ has a least fixed point $\mu\boldsymbol{f}$ in $\mathbb{R}^n_{\geq 0}$ i.e., $\mu\boldsymbol{f} = \boldsymbol{f}(\mu\boldsymbol{f})$ and, in addition, $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{y})$ implies $\mu\boldsymbol{f} \leq \boldsymbol{y}$. Moreover, the sequence $(\boldsymbol{\kappa}^{(k)}_{\boldsymbol{f}})_{k\in\mathbb{N}}$ with $\boldsymbol{\kappa}^{(0)}_{\boldsymbol{f}} := \boldsymbol{0}$, and $\boldsymbol{\kappa}^{(k+1)}_{\boldsymbol{f}} := \boldsymbol{f}(\boldsymbol{\kappa}^{(k)}_{\boldsymbol{f}}) = \boldsymbol{f}^{k+1}(\boldsymbol{0})$ is monotonically increasing with respect to $\leq$ (i.e. $\boldsymbol{\kappa}^{(k)}_{\boldsymbol{f}} \leq \boldsymbol{\kappa}^{(k+1)}_{\boldsymbol{f}}$) and converges to $\mu\boldsymbol{f}$.*

In the following we call $(\boldsymbol{\kappa}^{(k)}_{\boldsymbol{f}})_{k\in\mathbb{N}}$ the *Kleene sequence* of $\boldsymbol{f}(\boldsymbol{X})$, and drop the subscript whenever $\boldsymbol{f}$ is clear from the context. Similarly, we sometimes write $\boldsymbol{\mu}$ instead of $\mu\boldsymbol{f}$.

A variable X_i of an MSP $\boldsymbol{f}(\boldsymbol{X})$ is *productive* if $\kappa^{(k)}_i > 0$ for some $k \in \mathbb{N}$. An MSP is *clean* if all its variables are productive. It is easy to see that $\kappa^{(n)}_i = 0$ implies $\kappa^{(k)}_i = 0$ for all $k \in \mathbb{N}$. As for context-free grammars we can determine all productive variables in time linear in the size of $\boldsymbol{f}$.

Notation 2.4. In the following, we always assume that an MSP $\boldsymbol{f}$ is clean and feasible. I.e., whenever we write "MSP", we mean "clean and feasible MSP", unless explicitly stated otherwise.

For the formal definition of the *Decomposed Newton's Method (DNM)* (see also Section 1) we need the notion of *dependence* between variables.

Definition 2.5. Let $\boldsymbol{f}(\boldsymbol{X})$ be an MSP. X_i *depends directly* on X_k, denoted by $X_i \trianglelefteq X_k$, if $\frac{\partial f_i}{\partial X_k}(\boldsymbol{X})$ is not the zero-polynomial. X_i *depends* on X_k if $X_i \trianglelefteq^* X_k$, where $\trianglelefteq^*$ is the reflexive transitive closure of $\trianglelefteq$. An MSP is *strongly connected* (short: an *scMSP*) if all its variables depend on each other.

Any MSP can be decomposed into strongly connected components (SCCs), where an SCC S is a maximal set of variables such that each variable in S depends on each other variable in S. The following result for strongly connected MSPs was proved in [10, 12]:

Theorem 2.6. *Let $\boldsymbol{f}(\boldsymbol{X})$ be an scMSP and define the Newton operator $\mathcal{N}_{\boldsymbol{f}}$ as follows*

$$\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{X}) = \boldsymbol{X} + (\mathrm{Id} - \boldsymbol{f}'(\boldsymbol{X}))^{-1}(\boldsymbol{f}(\boldsymbol{X}) - \boldsymbol{X}) \; .$$

We have: (1) $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{x})$ is defined for all $\boldsymbol{0} \leq \boldsymbol{x} \prec \mu\boldsymbol{f}$ (i.e., $(\mathrm{Id} - \boldsymbol{f}'(\boldsymbol{x}))^{-1}$ exists). Moreover, $\boldsymbol{f}'(\boldsymbol{x})^* = \sum_{k\in\mathbb{N}} \boldsymbol{f}'(\boldsymbol{x})^k$ exists for all $\boldsymbol{0} \leq \boldsymbol{x} \prec \mu\boldsymbol{f}$, and so $\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{X}) = \boldsymbol{X} + \boldsymbol{f}'(\boldsymbol{X})^*(\boldsymbol{f}(\boldsymbol{X}) - \boldsymbol{X})$. (2) The Newton sequence $(\boldsymbol{\nu}_{\boldsymbol{f}}^{(k)})_{k\in\mathbb{N}}$ with $\boldsymbol{\nu}^{(k)} = \mathcal{N}_{\boldsymbol{f}}^k(\boldsymbol{0})$ is monotonically increasing, bounded from above by $\mu\boldsymbol{f}$ (i.e. $\boldsymbol{\nu}^{(k)} \leq \boldsymbol{f}(\boldsymbol{\nu}^{(k)}) \leq \boldsymbol{\nu}^{(k+1)} \prec \mu\boldsymbol{f}$), and converges to $\mu\boldsymbol{f}$.

DNM works by substituting the variables of lower SCCs by corresponding Newton approximations that were obtained earlier.

3. A Threshold for scMSPs

In this section we obtain a threshold after which DNM is guaranteed to converge linearly with rate 1.

We showed in [16] that for worst-case results on the convergence of Newton's method it is enough to consider *quadratic* MSPs, i.e., MSPs whose monomials have degree at most 2. The reason is that any MSP (resp. scMSP) $\boldsymbol{f}$ can be transformed into a quadratic MSP (resp. scMSP) $\widetilde{\boldsymbol{f}}$ by introducing auxiliary variables. This transformation is very similar to the transformation of a context-free grammar into Chomsky normal form. The transformation does not accelerate DNM, i.e., DNM on $\boldsymbol{f}$ is at least as fast (in a formal sense) as DNM on $\widetilde{\boldsymbol{f}}$, and so for a worst-case analysis, it suffices to consider quadratic systems. We refer the reader to [16] for details.

We start by defining the notion of "valid bits".

Definition 3.1. Let $\boldsymbol{f}(\boldsymbol{X})$ be an MSP. A vector $\boldsymbol{\nu}$ has i *valid bits* of the least fixed point $\mu\boldsymbol{f}$ if $\left|\mu\boldsymbol{f}_j - \nu_j\right| / \left|\mu\boldsymbol{f}_j\right| \leq 2^{-i}$ for every $1 \leq j \leq n$.

In the rest of the section we prove the following:

Theorem 3.2. *Let $\boldsymbol{f}(\boldsymbol{X})$ be a quadratic scMSP. Let c_{min} be the smallest nonzero coefficient of $\boldsymbol{f}$ and let μ_{min} and μ_{max} be the minimal and maximal component of $\mu\boldsymbol{f}$, respectively. Let*

$$k_{\boldsymbol{f}} = n \cdot \log\left(\frac{\mu_{max}}{c_{min} \cdot \mu_{min} \cdot \min\{\mu_{min}, 1\}}\right) .$$

Then $\boldsymbol{\nu}^{(\lceil k_{\boldsymbol{f}} \rceil + i)}$ has i valid bits of $\mu\boldsymbol{f}$ for every $i \geq 0$.

Loosely speaking, the theorem states that after $k_{\boldsymbol{f}}$ iterations of Newton's method, every subsequent iteration guarantees at least one more valid bit. It may be objected that $k_{\boldsymbol{f}}$ depends on the least fixed point $\mu\boldsymbol{f}$, which is precisely what Newton's method should compute. However, in the next section we show that there are important classes of MSPs (in fact, those which motivated our investigation), for which bounds on $\mu_{\min}$ can be easily obtained.

The following corollary is weaker than Theorem 3.2, but less technical in that it avoids a dependence on $\mu_{\max}$ and $c_{\min}$.

Corollary 3.3. *Let $\boldsymbol{f}(\boldsymbol{X})$ be a quadratic scMSP of dimension n whose coefficients are given as ratios of m-bit integers. Let μ_{min} be the minimal component of $\mu\boldsymbol{f}$. Let $k_{\boldsymbol{f}} = 3n^2m + 2n^2\,|\log \mu_{min}|$. Then $\boldsymbol{\nu}^{(\lceil k_{\boldsymbol{f}} \rceil + i)}$ has at least i valid bits of $\mu\boldsymbol{f}$ for every $i \geq 0$.*

Corollary 3.3 follows from Theorem 3.2 by a suitable bound on $\mu_{\max}$ in terms of $c_{\min}$ and $\mu_{\min}$ [5] (notice that, since $c_{\min}$ is the quotient of two m-bit integers, we have $c_{\min} \geq 1/2^m$).

In the rest of the section we sketch the proof of Theorem 3.2. The proof makes crucial use of vectors $\boldsymbol{d} \succ \boldsymbol{0}$ such that $\boldsymbol{d} \geq \boldsymbol{f}'(\mu\boldsymbol{f})\boldsymbol{d}$. We call a vector satisfying these two conditions a *cone vector of* $\boldsymbol{f}$ or, when $\boldsymbol{f}$ is clear from the context, just a cone vector.

In a previous paper we have shown that if the matrix $(\mathrm{Id} - \boldsymbol{f}'(\mu\boldsymbol{f}))$ is singular, then $\boldsymbol{f}$ has a cone vector ([16], Lemmata 4 and 8). As a first step towards the proof of Theorem 3.2 we show the following stronger proposition.

Proposition 3.4. *Any scMSP has a cone vector.*

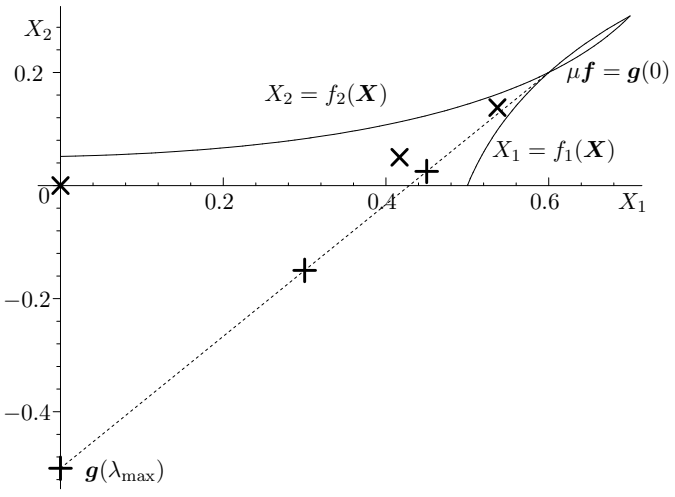
To a cone vector $\boldsymbol{d} = (d_1, \ldots, d_n)$ we associate two parameters, namely the maximum and the minimum of the ratios $\mu\boldsymbol{f}_1/d_1, \mu\boldsymbol{f}_2/d_2, \ldots, \mu\boldsymbol{f}_n/d_n$, which we denote by $\lambda_{\max}$ and $\lambda_{\min}$, respectively. The second step consists of showing (Proposition 3.6) that given a cone vector $\boldsymbol{d}$, the threshold $k_{\boldsymbol{f},\boldsymbol{d}} = \log(\lambda_{\max}/\lambda_{\min})$ satisfies the same property as $k_{\boldsymbol{f}}$ in Theorem 3.2, i.e., $\boldsymbol{\nu}^{(\lceil k_{\boldsymbol{f},\boldsymbol{d}} \rceil + i)}$ has i valid bits of $\mu\boldsymbol{f}$ for every $i \geq 0$. This follows rather easily from the following fundamental property of cone vectors: a cone vector leads to an upper bound on the error of Newton's method.

Lemma 3.5. *Let $\boldsymbol{d}$ be a cone vector of an MSP $\boldsymbol{f}$ and let $\lambda_{max} = \max\{\frac{\mu\boldsymbol{f}_i}{d_i}\}$. Then*

$$\mu\boldsymbol{f} - \boldsymbol{\nu}^{(k)} \leq 2^{-k}\lambda_{max}\,\boldsymbol{d}.$$

Proof Idea. Consider the ray $\boldsymbol{g}(t) = \mu\boldsymbol{f} - t\boldsymbol{d}$ starting in $\mu\boldsymbol{f}$ and headed in the direction $-\boldsymbol{d}$ (the dashed line in the picture below). It is easy to see that $\boldsymbol{g}(\lambda_{\max})$ is the intersection of $\boldsymbol{g}$ with an axis which is located farthest from $\mu\boldsymbol{f}$. One can then prove $\boldsymbol{g}(\frac{1}{2}\lambda_{\max}) \leq \boldsymbol{\nu}^{(1)}$, where $\boldsymbol{g}(\frac{1}{2}\lambda_{\max})$ is the point of the ray equidistant from $\boldsymbol{g}(\lambda_{\max})$ and $\mu\boldsymbol{f}$. By repeated application of this argument one obtains $\boldsymbol{g}(2^{-k}\lambda_{\max}) \leq \boldsymbol{\nu}^{(k)}$ for all $k \in \mathbb{N}$.

The following picture shows the Newton iterates $\boldsymbol{\nu}^{(k)}$ for $0 \leq k \leq 2$ (shape: $\times$) and the corresponding points $\boldsymbol{g}(2^{-k}\lambda_{\max})$ (shape: $+$) located on the ray $\boldsymbol{g}$. Notice that $\boldsymbol{\nu}^{(k)} \geq \boldsymbol{g}(2^{-k}\lambda_{\max})$. ∎



Now we easily obtain:

Proposition 3.6. *Let $\boldsymbol{f}(\boldsymbol{X})$ be an scMSP and let $\boldsymbol{d}$ be a cone vector of $\boldsymbol{f}$. Let $k_{\boldsymbol{f},\boldsymbol{d}} = \log \frac{\lambda_{max}}{\lambda_{min}}$, where $\lambda_{max} = \max_j \frac{\mu \boldsymbol{f}_j}{d_j}$ and $\lambda_{min} = \min_j \frac{\mu \boldsymbol{f}_j}{d_j}$. Then $\boldsymbol{\nu}^{(\lceil k_{\boldsymbol{f},\boldsymbol{d}} \rceil + i)}$ has at least i valid bits of $\mu \boldsymbol{f}$ for every $i \geq 0$.*

We now proceed to the third and final step. We have the problem that $k_{\boldsymbol{f},\boldsymbol{d}}$ depends on the cone vector $\boldsymbol{d}$, about which we only know that it exists (Proposition 3.4). We now sketch how to obtain the threshold $k_{\boldsymbol{f}}$ claimed in Theorem 3.2, which is independent of any cone vectors.

Consider Proposition 3.6 and let $\lambda_{\max} = \frac{\mu \boldsymbol{f}_i}{d_i}$ and $\lambda_{\min} = \frac{\mu \boldsymbol{f}_j}{d_j}$. We have $k_{\boldsymbol{f},\boldsymbol{d}} = \log\left(\frac{d_j}{d_i} \cdot \frac{\mu \boldsymbol{f}_i}{\mu \boldsymbol{f}_j}\right)$. The idea is to bound $k_{\boldsymbol{f},\boldsymbol{d}}$ in terms of $c_{\min}$. We show that if $k_{\boldsymbol{f},\boldsymbol{d}}$ is very large, then there must be variables X, Y such that X depends on Y only via a monomial that has a very small coefficient, which implies that $c_{\min}$ is very small.

4. Stochastic Models

As mentioned in the introduction, several problems concerning stochastic models can be reduced to problems about the least solution $\mu \boldsymbol{f}$ of an MSPE $\boldsymbol{f}$. In these cases, $\mu \boldsymbol{f}$ is a vector of probabilities, and so $\mu_{\max} \leq 1$. Moreover, we can obtain information on $\mu_{\min}$, which leads to bounds on the threshold $k_{\boldsymbol{f}}$.

4.1. Probabilistic Pushdown Automata

Our study of MSPs was initially motivated by the verification of probabilistic pushdown automata. A *probabilistic pushdown automaton (pPDA)* is a tuple $\mathcal{P} = (Q, \Gamma, \delta, Prob)$ 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 $pX \hookrightarrow q\alpha$ instead of $(p, X, q, \alpha) \in \delta$), and *Prob* is a function which to each transition $pX \hookrightarrow q\alpha$ assigns its probability $Prob(pX \hookrightarrow q\alpha) \in (0, 1]$ so that for all $p \in Q$ and $X \in \Gamma$ we have $\sum_{pX \hookrightarrow q\alpha} Prob(pX \hookrightarrow q\alpha) = 1$. We write $pX \overset{x}{\hookrightarrow} q\alpha$ instead of $Prob(pX \hookrightarrow q\alpha) = x$. A *configuration* of $\mathcal{P}$ is a pair qw, where q is a control state and $w \in \Gamma^*$ is a *stack content*. A probabilistic pushdown automaton $\mathcal{P}$ naturally induces a possibly infinite Markov chain with the configurations as states and transitions given by: $pX\beta \overset{x}{\hookrightarrow} q\alpha\beta$ for every $\beta \in \Gamma^*$ iff $pX \overset{x}{\hookrightarrow} q\alpha$. We assume w.l.o.g. that if $pX \overset{x}{\hookrightarrow} q\alpha$ is a transition then $|\alpha| \leq 2$.

pPDAs and the equivalent model of recursive Markov chains have been very thoroughly studied [6, 2, 10, 8, 7, 9, 11]. These papers have shown that the key to the analysis of pPDAs are the *termination probabilities* $[pXq]$, where p and q are states, and X is a stack letter, defined as follows (see e.g. [6] for a more formal definition): $[pXq]$ is the probability that, starting at the configuration pX, the pPDA eventually reaches the configuration $q\varepsilon$ (empty stack). It is not difficult to show that the vector of termination probabilities is the least fixed point of the MSPE containing the equation

$$[pXq] = \sum_{pX \overset{x}{\hookrightarrow} rYZ} x \cdot \sum_{t \in Q} [rYt] \cdot [tZq] \quad + \quad \sum_{pX \overset{x}{\hookrightarrow} rY} x \cdot [rYq] \quad + \quad \sum_{pX \overset{x}{\hookrightarrow} q\varepsilon} x$$

for each triple (p, X, q). Call this quadratic MSPE the *termination MSPE* of the pPDA (we assume that termination MSPEs are clean, and it is easy to see that they are always

feasible). We immediately have that if $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ is a termination MSP, then $\mu_{\max} \leq 1$. We also obtain a lower bound on $\mu_{\min}$:

Lemma 4.1. *Let $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$ be a termination MSPE with n variables. Then $\mu_{min} \geq c_{min}^{(2^{n+1}-1)}$.*

Together with Theorem 3.2 we get the following exponential bound for $k_{\boldsymbol{f}}$.

Proposition 4.2. *Let $\boldsymbol{f}$ be a strongly connected termination MSP with n variables and whose coefficients are expressed as ratios of m-bit numbers. Then $k_{\boldsymbol{f}} \leq n2^{n+2}m$.*

We conjecture that there is a lower bound on $k_{\boldsymbol{f}}$ which is exponential in n for the following reason. We know a family $(\boldsymbol{f}^{(n)})_{n=1,3,5,\ldots}$ of strongly connected MSPs with n variables and irrational coefficients such that $c_{\min}^{(n)} = \frac{1}{4}$ for all n and $\mu_{\min}^{(n)}$ is double-exponentially small in n. Experiments suggest that $\Theta(2^n)$ iterations are needed for the first bit of $\mu\boldsymbol{f}^{(n)}$, but we do not have a proof.

4.2. Strict pPDAs and Back-Button Processes

A pPDA is *strict* if for every $pX \in Q \times \Gamma$ the transition relation contains a pop-rule $pX \overset{x}{\hookrightarrow} q\epsilon$ for some $q \in Q$ and some $x > 0$. Essentially, strict pPDAs model programs in which every procedure has at least one terminating execution that does not call any other procedure. The termination MSP of a strict pPDA is of the form $\boldsymbol{b}(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{l}\boldsymbol{X} + \boldsymbol{c}$ for $\boldsymbol{c} \succ \boldsymbol{0}$. So we have $\mu\boldsymbol{f} \geq \boldsymbol{c}$, which implies $\mu_{\min} \geq c_{\min}$. Together with Theorem 3.2 we get:

Proposition 4.3. *Let $\boldsymbol{f}$ be a strongly connected termination MSP with n variables and whose coefficients are expressed as ratios of m-bit numbers. If $\boldsymbol{f}$ is derived from a strict pPDA, then $k_{\boldsymbol{f}} \leq 3nm$.*

Since in most applications m is small, we obtain an excellent convergence threshold.

In [13, 14] Fagin et al. introduce a special class of strict pPDAs called *back-button processes*: in a back-button process there is only one control state p , and any rule is of the form $pA \overset{b_A}{\hookrightarrow} p\varepsilon$ or $pA \overset{l_{AB}}{\hookrightarrow} pBA$. So the stack corresponds to a path through a finite graph with Γ as set of nodes and edges $A \to B$ for $pA \overset{l_{AB}}{\hookrightarrow} pBA$.

In [13, 14] back-button processes are used to model the behaviour of web-surfers: Γ is the set of web-pages, l_{AB} is the probability that a web-surfer uses a link from page A to page B, and b_A is the probability that the surfer pushes the "back"-button of the web-browser while visiting A. Thus, the termination probability $[pAp]$ is simply the probability that, if A is on top of the stack, A is eventually popped from the stack. The termination probabilities are the least solution of the MSPE consisting of the equations

$$[pAp] \quad = \quad b_A + \sum_{pA \overset{l_{AB}}{\hookrightarrow} pBA} l_{AB}[pBp][pAp] \quad = \quad b_A + [pAp] \sum_{pA \overset{l_{AB}}{\hookrightarrow} pBA} l_{AB}[pBp].$$

4.3. An Example

As an example of application of Theorem 3.2 consider the following scMSPE $\boldsymbol{X} = \boldsymbol{f}(\boldsymbol{X})$.

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 0.4X_2X_1 + 0.6 \\ 0.3X_1X_2 + 0.4X_3X_2 + 0.3 \\ 0.3X_1X_3 + 0.7 \end{pmatrix}$$

The least solution of the system gives the revocation probabilities of a back-button process with three web-pages. For instance, if the surfer is at page 2 it can choose between following links to pages 1 and 3 with probabilities 0.3 and 0.4, respectively, or pressing the back button with probability 0.3.

We wish to know if any of the revocation probabilities is equal to 1. Performing 14 Newton steps (e.g. with Maple) yields an approximation $\boldsymbol{\nu}^{(14)}$ to the termination probabilities with

$$\begin{pmatrix} 0.98 \\ 0.97 \\ 0.992 \end{pmatrix} \leq \boldsymbol{\nu}^{(14)} \leq \begin{pmatrix} 0.99 \\ 0.98 \\ 0.993 \end{pmatrix} .$$

We have $c_{\min} = 0.3$. In addition, since Newton's method converges to $\mu\boldsymbol{f}$ from below, we know $\mu_{\min} \geq 0.97$. Moreover, $\mu_{\max} \leq 1$, as $\mathbf{1} = \boldsymbol{f}(\mathbf{1})$ and so $\mu\boldsymbol{f} \leq \mathbf{1}$. Hence $k_{\boldsymbol{f}} \leq 3 \cdot \log \frac{1}{0.97 \cdot 0.3 \cdot 0.97} \leq 6$. Theorem 3.2 then implies that $\boldsymbol{\nu}^{(14)}$ has (at least) 8 valid bits of $\mu\boldsymbol{f}$. As $\mu\boldsymbol{f} \leq \mathbf{1}$, the absolute errors are bounded by the relative errors, and since $2^{-8} \leq 0.004$ we know:

$$\mu\boldsymbol{f} \prec \boldsymbol{\nu}^{(14)} + \begin{pmatrix} 2^{-8} \\ 2^{-8} \\ 2^{-8} \end{pmatrix} \prec \begin{pmatrix} 0.994 \\ 0.984 \\ 0.997 \end{pmatrix} \prec \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

So Theorem 3.2 gives a proof that all 3 revocation probabilities are strictly smaller than 1.

5. Linear Convergence of the Decomposed Newton's Method

Given a strongly connected MSP $\boldsymbol{f}$, Theorem 3.2 states that, if we have computed $k_{\boldsymbol{f}}$ preparatory iterations of Newton's method, then after i additional iterations we can be sure to have computed at least i bits of $\mu\boldsymbol{f}$. We call this linear convergence with rate 1. Now we show that DNM, which handles non-strongly-connected MSPs, converges linearly as well. We also give an explicit convergence rate.

Let $\boldsymbol{f}(\boldsymbol{X})$ be any quadratic MSP (again we assume *quadratic* MSPs throughout this section), and let $h(\boldsymbol{f})$ denote the height of the DAG of strongly connected components (SCCs). The convergence rate of DNM crucially depends on this height: In the worst case one needs asymptotically $\Theta(2^{h(\boldsymbol{f})})$ iterations in each component per bit, assuming one performs the same number of iterations in each component.

To get a sharper result, we suggest to perform a different number of iterations in each SCC, depending on its *depth*. The depth of an SCC S is the length of the longest path in the DAG of SCCs from S to a top SCC.

In addition, we use the following notation. For a depth t, we denote by $comp(t)$ the set of SCCs of depth t. Furthermore we define $C(t) := \bigcup comp(t)$ and $C_>(t) := \bigcup_{t'>t} C(t')$ and, analogously, $C_<(t)$. We will sometimes write $\boldsymbol{v}_t$ for $\boldsymbol{v}_{C(t)}$ and $\boldsymbol{v}_{>t}$ for $\boldsymbol{v}_{C_>(t)}$ and $\boldsymbol{v}_{<t}$ for $\boldsymbol{v}_{C_<(t)}$, where $\boldsymbol{v}$ is any vector.

Figure 1 shows the Decomposed Newton's Method (DNM) for computing an approximation $\boldsymbol{\nu}$ for $\mu\boldsymbol{f}$, where $\boldsymbol{f}(\boldsymbol{X})$ is any quadratic MSP. The authors of [10] recommend to

run Newton's Method in each SCC S until "approximate solutions for S are considered 'good enough' ". Here we suggest to run Newton's Method in each SCC S for a number of steps that depends (exponentially) on the depth of S and (linearly) on a parameter j that controls the number of iterations (see Figure 1).

function DNM $(\boldsymbol{f}, j)$
/* *The parameter j controls the number of iterations.* */
for t **from** $h(\boldsymbol{f})$ **downto** 0
  forall $S \in comp(t)$ /* *all SCCs S of depth t* */
    $\boldsymbol{\nu}_S := \mathcal{N}^{j\cdot 2^t}_{\boldsymbol{f}_S}(\mathbf{0})$ /* *$j \cdot 2^t$ iterations* */
    /* *apply $\boldsymbol{\nu}_S$ in the depending SCCs* */
    $\boldsymbol{f}_{<t}(\boldsymbol{X}) := \boldsymbol{f}_{<t}(\boldsymbol{X})[\boldsymbol{X}_S/\boldsymbol{\nu}_S]$
return $\boldsymbol{\nu}$

Figure 1: Decomposed Newton's Method (DNM) for computing an approximation $\boldsymbol{\nu}$ of $\mu\boldsymbol{f}$

Recall that $h(\boldsymbol{f})$ was defined as the height of the DAG of SCCs. Similarly we define the width $w(\boldsymbol{f})$ to be $\max_t |comp(t)|$. Notice that $\boldsymbol{f}$ has at most $(h(\boldsymbol{f})+1)\cdot w(\boldsymbol{f})$ SCCs. We have the following bound on the number of iterations run by DNM.

Proposition 5.1. *The function* DNM$(\boldsymbol{f}, j)$ *of Fig. 1 runs at most $j\cdot w(\boldsymbol{f})\cdot 2^{h(\boldsymbol{f})+1}$ iterations of Newton's method.*

We will now analyze the convergence behavior of DNM asymptotically (for large j). Let $\boldsymbol{\Delta}^{(j)}_S$ denote the error in S when running DNM with parameter j, i.e., $\boldsymbol{\Delta}^{(j)}_S := \boldsymbol{\mu}_S - \boldsymbol{\nu}^{(j)}_S$. Observe that the error $\boldsymbol{\Delta}^{(j)}_t$ can be understood as the sum of two errors:

$$\boldsymbol{\Delta}^{(j)}_t = \boldsymbol{\mu}_t - \boldsymbol{\nu}^{(j)}_t = (\boldsymbol{\mu}_t - \widetilde{\boldsymbol{\mu}}^{(j)}_t) + (\widetilde{\boldsymbol{\mu}}^{(j)}_t - \boldsymbol{\nu}^{(j)}_t)\,,$$

where $\widetilde{\boldsymbol{\mu}}^{(j)}_t := \mu\big(\boldsymbol{f}_t(\boldsymbol{X})[\boldsymbol{X}_{>t}/\boldsymbol{\nu}^{(j)}_{>t}]\big)$, i.e., $\widetilde{\boldsymbol{\mu}}^{(j)}_t$ is the least fixed point of $\boldsymbol{f}_t$ after the approximations from the lower SCCs have been applied. So, $\boldsymbol{\Delta}^{(j)}_t$ consists of the *propagation error* $(\boldsymbol{\mu}_t - \widetilde{\boldsymbol{\mu}}^{(j)}_t)$ and the newly inflicted *approximation error* $(\widetilde{\boldsymbol{\mu}}^{(j)}_t - \boldsymbol{\nu}^{(j)}_t)$.

The following lemma, technically non-trivial to prove, gives a bound on the propagation error.

Lemma 5.2 (Propagation error)**.** *Let $\boldsymbol{\nu}_{>t}$ be some approximation of $\boldsymbol{\mu}_{>t}$, i.e., $\mathbf{0} \le \boldsymbol{\nu}_{>t} \le \boldsymbol{\mu}_{>t}$. Let $\widetilde{\boldsymbol{\mu}_t} = \mu\big(\boldsymbol{f}_t(\boldsymbol{X})[\boldsymbol{X}_{>t}/\boldsymbol{\nu}_{>t}]\big)$. Then there is a constant $c > 0$ such that*

$$\|\boldsymbol{\mu}_t - \widetilde{\boldsymbol{\mu}_t}\| \le c \cdot \sqrt{\|\boldsymbol{\mu}_{>t} - \boldsymbol{\nu}_{>t}\|}\,.$$

Intuitively, Lemma 5.2 states that if $\boldsymbol{\nu}_{>t}$ has k valid bits of $\boldsymbol{\mu}_{>t}$, then $\widetilde{\boldsymbol{\mu}_t}$ has roughly $k/2$ valid bits of $\boldsymbol{\mu}_t$. In other words, (at most) one half of the valid bits are lost on each level of the DAG due to the propagation error.

The following theorem assures that after combining the propagation error and the approximation error, DNM still converges linearly.

Theorem 5.3. *Let $\boldsymbol{f}$ be a quadratic MSP. Let $\boldsymbol{\nu}^{(j)}$ denote the result of calling* DNM$(\boldsymbol{f}, j)$ *(see Figure 1). Then there is a $k_{\boldsymbol{f}} \in \mathbb{N}$ such that $\boldsymbol{\nu}^{(k_{\boldsymbol{f}}+i)}$ has at least i valid bits of $\mu\boldsymbol{f}$ for every $i \ge 0$.*

We conclude that increasing i by one gives us asymptotically at least one additional bit in each component and, by Proposition 5.1, costs $w(\boldsymbol{f}) \cdot 2^{h(\boldsymbol{f})+1}$ additional Newton iterations.

In the technical report [5] we give an example that shows that the bound above is essentially optimal in the sense that an exponential (in $h(\boldsymbol{f})$) number of iterations is in general needed to obtain an additional bit.

6. Newton's Method for General MSPs

Etessami and Yannakakis [10] introduced DNM because they could show that the matrix inverses used by Newton's method exist if Newton's method is run on each SCC separately (see Theorem 2.6).

It may be surprising that the matrix inverses used by Newton's method exist even if the MSP is *not* decomposed. More precisely one can show the following theorem, see [5].

Theorem 6.1. *Let $\boldsymbol{f}(\boldsymbol{X})$ be any MSP, not necessarily strongly connected. Let the Newton operator $\mathcal{N}_{\boldsymbol{f}}$ be defined as before:*

$$\mathcal{N}_{\boldsymbol{f}}(\boldsymbol{X}) = \boldsymbol{X} + (\mathrm{Id} - \boldsymbol{f}'(\boldsymbol{X}))^{-1}(\boldsymbol{f}(\boldsymbol{X}) - \boldsymbol{X})$$

Then the Newton sequence $(\boldsymbol{\nu}_{\boldsymbol{f}}^{(k)})_{k\in\mathbb{N}}$ with $\boldsymbol{\nu}^{(k)} = \mathcal{N}_{\boldsymbol{f}}^{k}(\mathbf{0})$ is well-defined (i.e., the matrix inverses exist), monotonically increasing, bounded from above by $\mu\boldsymbol{f}$ (i.e. $\boldsymbol{\nu}^{(k)} \leq \boldsymbol{\nu}^{(k+1)} \prec \mu\boldsymbol{f}$), and converges to $\mu\boldsymbol{f}$.

By exploiting Theorem 5.3 and Theorem 6.1 one can show the following theorem which addresses the convergence *speed* of Newton's Method in general.

Theorem 6.2. *Let $\boldsymbol{f}$ be any quadratic MSP. Then the Newton sequence $(\boldsymbol{\nu}^{(k)})_{k\in\mathbb{N}}$ is well-defined and converges linearly to $\mu\boldsymbol{f}$. More precisely, there is a $k_{\boldsymbol{f}} \in \mathbb{N}$ such that $\boldsymbol{\nu}^{(k_{\boldsymbol{f}}+i\cdot(h(\boldsymbol{f})+1)\cdot 2^{h(\boldsymbol{f})})}$ has at least i valid bits of $\mu\boldsymbol{f}$ for every $i \geq 0$.*

Again, the $2^{h(\boldsymbol{f})}$ factor cannot be avoided in general as shown by an example in [5].

7. Conclusions

We have proved a threshold $k_{\boldsymbol{f}}$ for strongly connected MSPEs. After $k_{\boldsymbol{f}}+i$ Newton iterations we have i bits of accuracy. The threshold $k_{\boldsymbol{f}}$ depends on the representation size of $\boldsymbol{f}$ and on the least solution $\mu\boldsymbol{f}$. Although this latter dependence might seem to be a problem, lower and upper bounds on $\mu\boldsymbol{f}$ can be easily derived for stochastic models (probabilistic programs with procedures, stochastic context-free grammars and back-button processes). In particular, this allows us to show that $k_{\boldsymbol{f}}$ depends linearly on the representation size for back-button processes. We have also shown by means of an example that the threshold $k_{\boldsymbol{f}}$ improves when the number of iterations increases.

In [16] we left the problem whether DNM converges linearly for non-strongly-connected MSPEs open. We have proven that this is the case, although the convergence rate is poorer: if h and w are the height and width of the graph of SCCs of $\boldsymbol{f}$, then there is a threshold $\widetilde{k}_{\boldsymbol{f}}$ such that $\widetilde{k}_{\boldsymbol{f}} + i \cdot w \cdot 2^{h+1}$ iterations of DNM compute at least i valid bits of $\mu\boldsymbol{f}$, where the exponential factor cannot be avoided in general.

Finally, we have shown that the Jacobian of the whole MSPE is guaranteed to exist, whether the MSPE is strongly connected or not.

Acknowledgments.

The authors wish to thank Kousha Etessami and anonymous referees for very valuable comments.

References

[1] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the Amer. Math. Society*, 21(1):1–46, 1989.

[2] T. Brázdil, A. Kučera, and O. Stražovský. On the decidability of temporal properties of probabilistic pushdown automata. In *Proceedings of STACS'2005*, volume 3404 of *LNCS*, pages 145–157. Springer, 2005.

[3] R.D. Dowell and S.R. Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5(71), 2004.

[4] R. Durbin, S.R. Eddy, A. Krogh, and G.J. Michison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[5] J. Esparza, S. Kiefer, and M. Luttenberger. Convergence thresholds of Newton's method for monotone polynomial equations. Technical report, Technische Universität München, 2007.

[6] J. Esparza, A. Kučera, and R. Mayr. Model-checking probabilistic pushdown automata. In *Proceedings of LICS 2004*, pages 12–21, 2004.

[7] J. Esparza, A. Kučera, and R. Mayr. Quantitative analysis of probabilistic pushdown automata: Expectations and variances. In *Proceedings of LICS 2005*, pages 117–126. IEEE Computer Society Press, 2005.

[8] K. Etessami and M. Yannakakis. Algorithmic verification of recursive probabilistic systems. In *Proceedings of TACAS 2005*, LNCS 3440, pages 253–270. Springer, 2005.

[9] K. Etessami and M. Yannakakis. Checking LTL properties of recursive Markov chains. In *Proceedings of 2nd Int. Conf. on Quantitative Evaluation of Systems (QEST'05)*, 2005.

[10] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. In *STACS*, pages 340–352, 2005.

[11] K. Etessami and M. Yannakakis. Recursive Markov decision processes and recursive stochastic games. In *Proceedings of ICALP 2005*, volume 3580 of *LNCS*, pages 891–903. Springer, 2005.

[12] K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations, 2006. Draft journal submission, `http://homepages.inf.ed.ac.uk/kousha/bib_index.html`.

[13] R. Fagin, A.R. Karlin, J. Kleinberg, P. Raghavan, S. Rajagopalan, R. Rubinfeld, M. Sudan, and A. Tomkins. Random walks with "back buttons" (extended abstract). In *STOC*, pages 484–493, 2000.

[14] R. Fagin, A.R. Karlin, J. Kleinberg, P. Raghavan, S. Rajagopalan, R. Rubinfeld, M. Sudan, and A. Tomkins. Random walks with "back buttons". *Annals of Applied Probability*, 11(3):810–862, 2001.

[15] S. Geman and M. Johnson. Probabilistic grammars and their applications, 2002.

[16] S. Kiefer, M. Luttenberger, and J. Esparza. On the convergence of Newton's method for monotone systems of polynomial equations. In *Proceedings of STOC*, pages 217–226. ACM, 2007.

[17] B. Knudsen and J. Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Research*, 31(13):3423–3428, 2003.

[18] W. Kuich. *Handbook of Formal Languages*, volume 1, chapter 9: Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata, pages 609 – 677. Springer, 1997.

[19] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[20] J.M. Ortega and W.C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. Academic Press, 1970.

[21] Y. Sakabikara, M. Brown, R. Hughey, I.S. Mian, K. Sjolander, R.C. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA. *Nucleic Acids Research*, 22:5112–5120, 1994.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 301-312
www.stacs-conf.org

MODEL CHECKING GAMES FOR THE QUANTITATIVE μ-CALCULUS

DIANA FISCHER [1], ERICH GRÄDEL [1], AND ŁUKASZ KAISER [1]

[1] Mathematische Grundlagen der Informatik, RWTH Aachen
E-mail address: {fischer,graedel,kaiser}@logic.rwth-aachen.de

Abstract. We investigate quantitative extensions of modal logic and the modal μ-calculus, and study the question whether the tight connection between logic and games can be lifted from the qualitative logics to their quantitative counterparts. It turns out that, if the quantitative μ-calculus is defined in an appropriate way respecting the duality properties between the logical operators, then its model checking problem can indeed be characterised by a quantitative variant of parity games. However, these quantitative games have quite different properties than their classical counterparts, in particular they are, in general, not positionally determined. The correspondence between the logic and the games goes both ways: the value of a formula on a quantitative transition system coincides with the value of the associated quantitative game, and conversely, the values of quantitative parity games are definable in the quantitative μ-calculus.

1. Introduction

There have been a number of recent proposals to extend the common qualitative, i.e. two-valued, logical formalisms for specifying the behaviour of concurrent systems, such as propositional modal logic ML, the temporal logics LTL and CTL, and the modal μ-calculus L_μ, to quantitative formalisms. In quantitative logics, the formulae can take, at a given state of a system, not just the values *true* and *false*, but quantitative values, for instance from the (non-negative) real numbers. There are several scenarios and applications where it is desirable to replace purely qualitative statements by quantitative ones, which can be of very different nature: we may be interested in the probability of an event, the value that we assign to an event may depend on how late it occurs, we can ask for the number of occurrences of an event in a play, and so on. We can consider transition structures, where already the atomic propositions take numeric values, or we can ask about the 'degree of satisfaction' of a property. There are several papers that deal with either of these topics, resulting in different specification formalisms and in different notions of transition structures. In particular, due to the prominence and importance of the modal μ-calculus in verification, there have been several attempts to define a quantitative μ-calculus. In some of these, the term quantitative refers to probability, i.e. the logic is interpreted over probabilistic transition systems [11], or used to describe winning conditions in stochastic games [5, 1, 8]. Other variants introduce quantities by allowing discounting in the respective version of

Key words and phrases: games, logic, model checking, quantitative logics.

a "next"-operator for qualitative transition systems [1], Markov decision processes and Markov chains [2], and for stochastic games [4].

While there certainly is ample motivation to extend qualitative specification formalisms to quantitative ones, there also are problems. As has been observed in many areas of mathematics, engineering and computer science where logical formalisms are applied, quantitative formalisms in general lack the clean and clear mathematical theory of their qualitative counterparts, and many of the desirable mathematical and algorithmic properties tend to get lost. Also, the definitions of quantitative formalisms are often ad hoc and do not always respect the properties that are required for logical methodologies. In this paper we have a closer look at quantitative modal logic and the quantitative μ-calculus in terms of their description by appropriate semantic games. The close connection to games is a fundamental aspect of logics. The evaluation of logical formulae can be described by model checking games, played by two players on an arena which is formed as the product of a structure $\mathcal{K}$ and a formula ψ. One player (Verifier) attempts to prove that ψ is satisfied in $\mathcal{K}$ while the other (Falsifier) tries to refute this.

For the modal μ-calculus L_μ, model checking is described by *parity games*, and this connection is of crucial importance for the model theory, the algorithmic evaluation and the applications of the μ-calculus. Indeed, most competitive model checking algorithms for L_μ are based on algorithms to solve the strategy problem in parity games [10]. Furthermore, parity games enjoy nice properties like positional determinacy and can be intuitively understood: often, the best way to make sense of a μ-calculus formula is to look at the associated game. In the other direction, winning regions of parity games (for any fixed number of priorities) are definable in the modal μ-calculus.

In this paper, we explore the question to what extent the relationship between the μ-calculus and parity games can be extended to a quantitative μ-calculus and appropriate quantitative model checking games. The extension is not straightforward, and requires that one defines the quantitative μ-calculus in the 'right' way, so as to ensure that it has appropriate closure and duality properties (such as closure under negation, De Morgan equalities, quantifier and fixed point dualities) to make it amenable to a game-based approach. Once this is done, we can indeed construct a quantitative variant of parity games, and prove that they are the appropriate model checking games for the quantitative μ-calculus. As in the classical setting the correspondence goes both ways: the value of a formula in a structure coincides with the value of the associated model checking game, and conversely, the values of quantitative parity games (with a fixed number of priorities) are definable in the quantitative μ-calculus. However, the mathematical properties of quantitative parity games are different from their qualitative counterparts. In particular, they are, in general, not positionally determined, not even up to approximation. The proof that the quantitative model checking games correctly describe the value of the formulae is considerably more difficult than for the classical case.

As in the classical case, model checking games lead to a better understanding of the semantics and expressive power of the quantitative μ-calculus. Further, the game-based approach also sheds light on the consequences of different choices in the design of the quantitative formalism, which are far less obvious than for classical logics.

2. Quantitative μ-calculus

In [3], de Alfaro, Faella, and Stoelinga introduce a quantitative μ-calculus, that is interpreted over metric transition systems, where predicates can take values in arbitrary metric spaces. Furthermore, their μ-calculus allows discounting in modalities and is studied in connection with quantitative versions of basic system relations such as bisimulation.

We base our calculus on the one proposed in [3] but modify it in the following ways.

(1) We decouple discounts from the modal operators.
(2) We allow discount factors to be greater than one.
(3) In the definition of transition systems we allow additional discounts on the edges.

These changes make the logic more robust and more general, and, as we will show in the next section, will permit us to introduce a negation operator with the desired duality properties that are fundamental to a game-based analysis.

Quantitative transition systems, similar to the ones introduced in [3] are directed graphs equipped with quantities at states and discounts on edges. In the sequel, $\mathbb{R}^+$ is the set of non-negative real numbers, and $\mathbb{R}^+_\infty := \mathbb{R}^+ \cup \{\infty\}$.

Definition 2.1. A *quantitative transition system (QTS)* is a tuple

$$\mathcal{K} = (V, E, \delta, \{P_i\}_{i\in I}),$$

consisting of a directed graph (V, E), a discount function $\delta : E \to \mathbb{R}^+ \setminus \{0\}$ and functions $P_i : V \to \mathbb{R}^+_\infty$, that assign to each state the values of the predicates at that state.

A transition system is *qualitative* if all functions P_i assign only the values 0 or ∞, i.e. $P_i : V \to \{0, \infty\}$, where 0 stands for false and ∞ for true, and it is *non-discounted* if $\delta(e) = 1$ for all $e \in E$.

We now introduce a quantitative version of the modal μ-calculus to describe properties of quantitative transition systems.

Definition 2.2. Given a set $\mathcal{V}$ of variables X, predicate functions $\{P_i\}_{i\in I}$, discount factors $d \in \mathbb{R}^+$ and constants $c \in \mathbb{R}^+$, the formulae of *quantitative μ-calculus* (Qμ) can be built in the following way:

(1) $|P_i - c|$ is a Qμ-formula,
(2) X is a Qμ-formula,
(3) if φ, ψ are Qμ-formulae, then so are $(\varphi \wedge \psi)$ and $(\varphi \vee \psi)$,
(4) if φ is a Qμ-formula, then so are $\Box\varphi$ and $\Diamond\varphi$,
(5) if φ is a Qμ-formula, then so is $d \cdot \varphi$,
(6) if φ is a formula of Qμ, then $\mu X.\varphi$ and $\nu X.\varphi$ are formulae of Qμ.

Formulae of Qμ are interpreted over quantitative transition systems. Let $\mathcal{F}$ be the set of functions $f : V \to \mathbb{R}^+_\infty$, with $f_1 \leq f_2$ if $f_1(v) \leq f_2(v)$ for all v. Then $(\mathcal{F}, \leq)$ forms a complete lattice with the constant functions $f = \infty$ as top element and $f = 0$ as bottom element.

Given an interpretation $\varepsilon : \mathcal{V} \to \mathcal{F}$, a variable $X \in \mathcal{V}$, and a function $f \in \mathcal{F}$, we denote by $\varepsilon[X \leftarrow f]$ the interpretation ε', such that $\varepsilon'(X) = f$ and $\varepsilon'(Y) = \varepsilon(Y)$ for all $Y \neq X$.

Definition 2.3. Given a QTS $\mathcal{K} = (V, E, \delta, \{P_i\}_{i\in I})$ and an interpretation ε, a Qμ-formula yields a valuation function $[\![\varphi]\!]^{\mathcal{K}}_\varepsilon : V \to \mathbb{R}^+_\infty$ defined as follows:

(1) $[\![|P_i - c|]\!]^{\mathcal{K}}_\varepsilon(v) = |P_i(v) - c|$,
(2) $[\![\varphi_1 \wedge \varphi_2]\!]^{\mathcal{K}}_\varepsilon = \min\{[\![\varphi_1]\!]^{\mathcal{K}}_\varepsilon, [\![\varphi_2]\!]^{\mathcal{K}}_\varepsilon\}$ and $[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{K}}_\varepsilon = \max\{[\![\varphi_1]\!]^{\mathcal{K}}_\varepsilon, [\![\varphi_2]\!]^{\mathcal{K}}\}$,

(3) $[\![\Diamond\varphi]\!]^{\mathcal{K}}_{\varepsilon}(v) = \sup_{v' \in vE} \delta(v,v') \cdot [\![\varphi]\!]^{\mathcal{K}}_{\varepsilon}(v')$ and $[\![\Box\varphi]\!]^{\mathcal{K}}_{\varepsilon}(v) = \inf_{v' \in vE} \frac{1}{\delta(v,v')} [\![\varphi]\!]^{\mathcal{K}}_{\varepsilon}(v')$,
(4) $[\![d \cdot \varphi]\!]^{\mathcal{K}}_{\varepsilon}(v) = d \cdot [\![\varphi]\!]^{\mathcal{K}}_{\varepsilon}(v)$,
(5) $[\![X]\!]^{\mathcal{K}}_{\varepsilon} = \varepsilon(X)$,
(6) $[\![\mu X.\varphi]\!]^{\mathcal{K}}_{\varepsilon} = \inf\{f \in \mathcal{F} : f = [\![\varphi]\!]^{\mathcal{K}}_{\varepsilon[X \leftarrow f]}\}$,
(7) $[\![\nu X.\varphi]\!]^{\mathcal{K}}_{\varepsilon} = \sup\{f \in \mathcal{F} : f = [\![\varphi]\!]^{\mathcal{K}}_{\varepsilon[X \leftarrow f]}\}$.

For formulae without free variables, we can simply write $[\![\varphi]\!]^{\mathcal{K}}$ rather than $[\![\varphi]\!]^{\mathcal{K}}_{\varepsilon}$.

We call the fragment of Qμ consisting of formulae without fixed-point operators *quantitative modal logic* QML. If Qμ is interpreted over qualitative transition systems, it coincides with the classical μ-calculus and we say that $\mathcal{K}, v$ is a model of φ, $\mathcal{K}, v \models \varphi$ if $[\![\varphi]\!]^{\mathcal{K}}(v) = \infty$. Over non-discounted quantitative transition systems, the definition above coincides with the one in [3]. For discounted systems we take the natural definition for $\Diamond$ and use the dual one for $\Box$, thus the $\frac{1}{\delta}$ factor. As we will show, this is the only definition for which there is a well-behaved negation operator and with a close relation to model checking games.

We always assume the formulae to be *well-named*, i.e. each fixed-point variable is bound only once and no variable appears both free and bound and we use the notions of *alternation level* and *alternation depth* in the usual way, as defined in e.g. [9].

Note that all operators in Qμ are monotone, thus guaranteeing the existence of the least and greatest fixed points, and their inductive definition according to the Knaster-Tarski Theorem stated below.

Proposition 2.4. *The least and greatest fixed points exist and can be computed inductively:* $[\![\mu X.\varphi]\!]^{\mathcal{K}}_{\varepsilon} = g_\gamma$ *with* $g_0(v) = 0$ *(and* $[\![\nu X.\varphi]\!]^{\mathcal{K}}_{\varepsilon} = g_\gamma$ *with* $g_0(v) = \infty$*) for all* $v \in V$ *where*

$$g_\alpha = \begin{cases} [\![\varphi]\!]_{\varepsilon[X \leftarrow g_{\alpha-1}]} & \text{for } \alpha \text{ successor ordinal,} \\ \lim_{\beta<\alpha} [\![\varphi]\!]_{\varepsilon[X \leftarrow g_\beta]} & \text{for } \alpha \text{ limit ordinal,} \end{cases}$$

and γ *is such that* $g_\gamma = g_{\gamma+1}$.

3. Negation and Duality

So far, the quantitative logics Qμ and QML lack a negation operator and the associated dualities between $\wedge$ and $\vee$, $\Diamond$ and $\Box$, and between least and greatest fixed points. Let us clarify in the following definition what we expect from such an operator.

Definition 3.1. A *negation operator* $f_\neg$ for Qμ is a function $\mathbb{R}^+_\infty \to \mathbb{R}^+_\infty$, such that when we define $[\![\neg\varphi]\!] = f_\neg([\![\varphi]\!])$, the following equivalences hold for every $\varphi \in$ Qμ:

(1) $\neg\neg\varphi \equiv \varphi$
(2) $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$ and $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$
(3) $\neg\Box\varphi \equiv \Diamond\neg\varphi$ and $\neg\Diamond\varphi \equiv \Box\neg\varphi$
(4) $\neg d \cdot \varphi \equiv \beta(d) \cdot \neg\varphi$ for some β independent of φ
(5) $\neg\mu X.\varphi \equiv \nu X.\neg\varphi[X/\neg X]$ and $\neg\nu X.\varphi \equiv \mu X.\neg\varphi[X/\neg X]$

A straightforward calculation shows that the function

$$f_{\frac{1}{x}} : \mathbb{R}^+_\infty \to \mathbb{R}^+_\infty : x \mapsto \begin{cases} 1/x & \text{for } x \neq 0, x \neq \infty, \\ \infty & \text{for } x = 0, \\ 0 & \text{for } x = \infty, \end{cases}$$

is a negation operator for Qμ. Hence, we can safely include negation into the definition of Qμ. If we do so, we of course have to demand that the fixed-point variables in the

definition of least and greatest fixed point formulae, see Definition 2.2, only occur under an even number of negations, so as to preserve monotonicity.

Moreover, we show that $f_{\frac{1}{x}}$ is the only negation operator with the required properties. You should note that this is the case even for non-discounted transition systems, and thus it motivates our definition of the semantics of Qμ, in particular of the modal operators, on quantitative transition systems.

Theorem 3.2. *$f_{\frac{1}{x}}$ is the only negation operator for Qμ, even for non-discounted systems.*

4. Quantitative Parity Games

Quantitative parity games are an extension of classical parity games. The two main differences are the possibility to assign real values in final positions to denote the payoff for Player 0 and the possibility to discount payoff values on edges.

Definition 4.1. A *quantitative parity game* is a tuple $\mathcal{G} = (V, V_0, V_1, E, \delta, \lambda, \Omega)$ where V is a disjoint union of V_0 and V_1, i.e. positions belong to either Player 0 or 1. The transition relation $E \subseteq V \times V$ describes possible moves in the game and $\delta : V \times V \to \mathbb{R}^+$ maps every move to a positive real value representing the discount factor. The payoff function $\lambda : \{v \in V : vE = \emptyset\} \to \mathbb{R}^+_\infty$ assigns values to all terminal positions and the priority function $\Omega : V \to \{0, \dots, n\}$ assigns a priority to every position.

How to play. Every play starts at some vertex $v \in V$. For every vertex in V_i, Player i chooses a successor vertex, and the play proceeds from that vertex. If the play reaches a terminal vertex, it ends. We denote by $\pi = v_0 v_1 \dots$ the (possibly infinite) play through vertices $v_0 v_1 \dots$, given that $(v_n, v_{n+1}) \in E$ for every n. The outcome $p(\pi)$ of a finite play $\pi = v_0 \dots v_k$ can be computed by multiplying all discount factors seen throughout the play with the value of the final node,

$$p(v_0 v_1 \dots v_k) = \delta(v_0, v_1) \cdot \delta(v_1, v_2) \cdot \dots \cdot \delta(v_{k-1}, v_k) \cdot \lambda(v_k).$$

The outcome of an infinite play depends only on the lowest priority seen infinitely often. We will assign the value 0 to every infinite play, where the lowest priority seen infinitely often is odd, and ∞ to those, where it is even.

Goals. The two players have opposing objectives regarding the outcome of the play. Player 0 wants to maximise the outcome, while Player 1 wants to minimise it.

Strategies. A strategy for player $i \in 0, 1$ is a function $s : V^* V_i \to V$ with $(v, s(v)) \in E$. A play $\pi = v_0 v_1 \dots$ is *consistent with a strategy* s for player i, if $v_{n+1} = s(v_0 \dots v_n)$ for every n such that $v_n \in V_i$. For strategies σ, ρ for the two players, we denote by $\pi_{\sigma,\rho}(v)$ the unique play starting at node v which is consistent with both σ and ρ.

Determinacy. A game is *determined* if, for each position v, the highest outcome Player 0 can assure from this position and the lowest outcome Player 1 can assure converge,

$$\sup_{\sigma \in \Gamma_0} \inf_{\rho \in \Gamma_1} p(\pi_{\sigma,\rho}(v)) = \inf_{\rho \in \Gamma_1} \sup_{\sigma \in \Gamma_0} p(\pi_{\sigma,\rho}(v)) =: \mathrm{val}\mathcal{G}(v),$$

where Γ_0, Γ_1 are the sets of all possible strategies for Player 0, Player 1 and the achieved outcome is called the *value of $\mathcal{G}$ at v*.

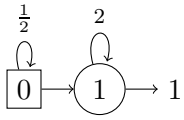
Classical parity games can be seen as a special case of quantitative parity games when we map winning to payoff ∞ and losing to payoff 0. Formally, we say that a quantitative parity game $\mathcal{G} = (V, V_0, V_1, E, \delta, \lambda, \Omega)$ is *qualitative* when $\lambda(v) = 0$ or $\lambda(v) = \infty$ for all $v \in V$

with $vE = \emptyset$. In qualitative games, we denote by $W_i \in V$ the winning region of player i, i.e. W_0 is the region where player 0 has a strategy to guarantee payoff ∞ and W_1 is the region where player 1 can guarantee payoff 0. Note that there is no need for the discount function δ in the qualitative case as the payoff can not be changed by discounting.

Qualitative parity games have been extensively studied in the past. One of their fundamental properties is *positional determinacy*. In every parity game, the set of positions can be partitioned into the winning regions W_0 and W_1 for the two players, and each player has a positional winning strategy on her winning region (which means that the moves selected by the strategy only depend on the current position, not on the history of the play).

Unfortunately, this result does not generalise to quantitative parity games. Example 4.2 shows that there are simple quantitative games where no player has a positional winning strategy. In the depicted game there is no optimal strategy for Player 0, and even if one fixes an approximation of the game value, Player 0 needs infinite memory to reach this approximation, because she needs to loop in the second position as long as Player 1 looped in the first one to make up for the discounts. (By convention, we depict positions of Player 0 with a circle and of Player 1 with a square and the number inside is the priority for non-terminal positions and the payoff in terminal ones.)

Example 4.2.



4.1. Model Checking Games for Qμ

A game $(\mathcal{G}, v)$ is a model checking game for a formula φ and a structure $\mathcal{K}, v'$, if the value of the game starting from v is exactly the value of the formula evaluated on $\mathcal{K}$ at v'. In the qualitative case, that means, that φ holds in $\mathcal{K}, v'$ if Player 0 wins in $\mathcal{G}$ from v.

Definition 4.3. For a quantitative transition system $\mathcal{K} = (S, T, \delta_S, P_i)$ and a Qμ-formula φ in negation normal form, the quantitative parity game $\mathrm{MC}[\mathcal{K}, \varphi] = (V, V_0, V_1, E, \delta, \lambda, \Omega)$, which we call the *model checking game* for $\mathcal{K}$ and φ, is constructed in the following way.
Positions. The positions of the game are the pairs (ψ, s), where ψ is a subformula of φ, and $s \in S$ is a state of the QTS $\mathcal{K}$, and the two special positions (0) and (∞). Positions (ψ, s) where the top operator of ψ is $\Box, \wedge$, or ν belong to Player 1 and all other positions belong to Player 0.
Moves. Positions of the form $(|P_i - c|, s), (0)$, and (∞) are terminal positions. From positions of the form $(\psi \wedge \theta, s)$, resp. $(\psi \vee \theta, s)$, one can move to (ψ, s) or to (θ, s). Positions of the form $(\Diamond\psi, s)$ have either a single successor (0), in case s is a terminal state in $\mathcal{K}$, or one successor (ψ, s') for every $s' \in sT$. Analogously, positions of the form $(\Box\psi, s)$ have a single successor (∞), if $sT = \emptyset$, or one successor (ψ, s') for every $s' \in sT$ otherwise. Positions of the form $(d \cdot \psi, s)$ have a unique successor (ψ, s'). Fixed-point positions $(\mu X.\psi, s)$, resp. $(\nu X.\psi, s)$ have a single successor (ψ, s). Whenever one encounters a position where the fixed-point variable stands alone, i.e. (X, s'), the play goes back to the corresponding definition, namely (ψ, s').
Discounts. The discount of an edge is d for transitions from positions $(d \cdot \psi, s)$, it is $\delta_S(s, s')$ for transitions from $(\Diamond\psi, s)$ to (ψ, s'), it is $1/\delta_S(s, s')$ for transitions from $(\Box\psi, s)$ to (ψ, s'), and 1 for all outgoing transitions from other positions.

Payoffs. The payoff function λ assigns $|[\![P_i]\!](s) - c|$ to all positions $(|P_i - c|, s)$, ∞ to position (∞), and 0 to position (0).
Priorities. The priority function Ω is defined as in the classical case using the alternation level of the fixed-point variables, see e.g. [9]. Positions (X, s) get a lower priority than positions (X', s') if X has a lower alternation level than X'. The priorities are then adjusted to have the right parity, so that an even value is assigned to all positions (X, s) where X is a ν-variable and an odd value to those where X is a μ-variable. The maximum priority, equal to the alternation depth of the formula, is assigned to all other positions.

It is well-known that qualitative parity games are model checking games for the classical μ-calculus, see e.g. [6] or [12]. A proof that uses the unfolding technique can be found in [9]. We generalise this connection to the quantitative setting as follows.

Theorem 4.4. *For every formula φ in $\mathrm{Q}\mu$, a quantitative transition system $\mathcal{K}$, and $v \in \mathcal{K}$, the game $\mathrm{MC}[\mathcal{K}, \varphi]$ is determined and*

$$\mathrm{valMC}[\mathcal{K}, \varphi](\varphi, v) = [\![\varphi]\!]^{\mathcal{K}}(v).$$

4.2. Unfolding Quantitative Parity Games

To prove the model checking theorem in the quantitative case, we start with games with one priority, known as reachability and safety games. The construction of ε-optimal strategies is obtained by a generalisation of backwards induction. At first, we fix the notation and show a few basic properties.

Definition 4.5. A number $k \in \mathbb{R}^+_\infty$ is called *ε-close* to $p \in \mathbb{R}^+_\infty$, when either p is finite and $|k-p| \leq \varepsilon$ or $p = \infty$ and $k \geq \frac{1}{\varepsilon}$. A strategy σ in a determined game $\mathcal{G}$ is *ε-optimal* from v if it assures a payoff ε-close to $\mathrm{val}\mathcal{G}(v)$. Furthermore, we say that k is *ε-above p* (or *ε-below*), if $k \geq p'$ (or $k \leq p'$) for some p' that is ε-close to p.

We slightly abuse the word "close" as ε-closeness is *not* symmetric, since $\frac{1}{\varepsilon}$ is ε-close to ∞, but ∞ is not ε-close to any number $r \in \mathbb{R}^+$. Still, the following lemmas should convince you that our definition suits our considerations well.

Definition 4.6. For every history $h = v_0 \dots v_\ell$ of a play, let $\Delta(h) = \Pi_{i<\ell}\delta(v_i, v_{i+1})$ be the product of all discount factors seen in h, and let $D(h) = \max(\Delta(h), \frac{1}{\Delta(h)})$. Note that for every play $\pi = v_0 v_1 \dots$ and every k,

$$p(\pi) = \Delta(v_0 \dots v_k) \cdot p(v_k v_{k+1} \dots).$$

Lemma 4.7. *Let $x, y \in \mathbb{R}^+_\infty$, $\varepsilon \in (0,1)$, $\Delta \in \mathbb{R}^+ \setminus \{0\}$, and $D = \max\{\Delta, \frac{1}{\Delta}\}$.*

(1) If x is ε/D-close to y, then $\Delta \cdot x$ is ε-close to $\Delta \cdot y$. This holds in particular when $\Delta = \Delta(h)$ and $D = D(h)$ for a history h.
(2) If x is $\varepsilon/2$-close to y and y is $\varepsilon/2$-close to z, then x is ε-close to z.

This lemma remains valid if we replace the close-relation by the above- or below-relation.

Proposition 4.8. *Reachability and Safety games are determined, for every position v there exist strategies σ^ε and ρ^ε that guarantee payoffs ε-above (or respectively ε-below)* $\mathrm{val}\mathcal{G}(v)$.

The next step is to prove the determinacy of quantitative parity games. For this purpose, we present a method to unfold a quantitative parity game into a sequence of games with a smaller number of priorities. This technique is inspired by the proof of correctness

of the model checking games for L_μ in [9]. We can extend this method to prove Theorem 4.4 by showing that, as in the classical case, the unfolding of $\mathrm{MC}[\mathcal{K}, \varphi]$ is closely related to the inductive evaluation of fixed points in φ on $\mathcal{K}$.

From now on, we assume that the minimal priority in $\mathcal{G}$ is even and call it m. This is no restriction, since, if the minimal priority is odd, we can always consider the dual game, where the roles of the players are switched and all priorities are decreased by one.

Definition 4.9. We define the *truncated game* $\mathcal{G}^- = (V, E^-, \lambda, \Omega^-)$ for a quantitative parity game $\mathcal{G} = (V, E, \lambda, \Omega)$. We assume without loss of generality that all nodes with minimal priority in $\mathcal{G}$ have unique successors with a discount of 1. In $\mathcal{G}^-$ we remove the outgoing edge from each of these nodes. Since these nodes are terminal positions in $\mathcal{G}^-$, their priority does not matter any more for the outcome of a play and Ω^- assigns them a higher priority, e.g. $m+1$. Formally,

$$E^- = E \setminus \{(v, v') : \Omega(v) = m\}$$

$$\Omega^-(v) = \begin{cases} \Omega(v) & \text{if } \Omega(v) \neq m, \\ m+1 & \text{if } \Omega(v) = m. \end{cases}$$

The *unfolding of* $\mathcal{G}$ is a sequence of games $\mathcal{G}^-_\alpha$, for ordinals α, which all coincide with $\mathcal{G}^-$, except for the valuation functions λ_α. Below we give the construction of the $\lambda_\alpha{}'s$.

For all terminal nodes v of the original game $\mathcal{G}$ we have $\lambda_\alpha(v) = \lambda(v)$ for all α. For the new terminal nodes, i.e. all $v \in V$, such that $vE^- = \emptyset$ and $vE = \{w\}$, the valuation is given by:

$$\lambda_\alpha(v) = \begin{cases} \infty & \text{for } \alpha = 0, \\ \mathrm{val}\mathcal{G}^-_{\alpha-1}(w) & \text{for } \alpha \text{ successor ordinal}, \\ \lim_{\beta<\alpha} \mathrm{val}\mathcal{G}^-_\beta(w) & \text{for } \alpha \text{ limit ordinal}. \end{cases}$$

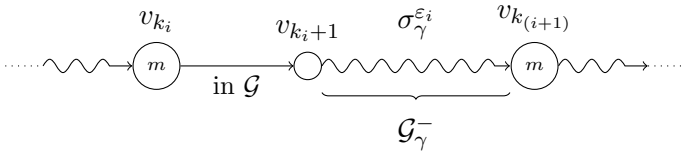
The intuition behind the definition of λ_α is to give an incentive for Player 0 to reach the new terminal nodes by first giving them the best possible valuation, and later by updating them to values of their successor in a previous game $\mathcal{G}^-_\beta, \beta < \alpha$.

To determine the value of the original game $\mathcal{G}$, we inductively compute the values for each game in $\mathcal{G}_\alpha$, until they do not change any more. Let γ be an ordinal for which $\mathrm{val}\mathcal{G}^-_\gamma = \mathrm{val}\mathcal{G}^-_{\gamma+1}$. Such an ordinal exists, since the values of the games in the unfolding are monotonically decreasing (which follows from determinacy of these games and definition). We set $g(v) = g_\gamma(v) = \mathrm{val}\mathcal{G}^-_\gamma(v)$ and show that g is the value function of the original game $\mathcal{G}$.

To prove this, we need to introduce strategies for Player 1 and Player 0, which are inductively constructed from the strategies in the unfolding. To give an intuition for the construction, we view a play in $\mathcal{G}$ as a play in the unfolding of $\mathcal{G}$. Let us look more closely at the situation of each player.

The Strategy of Player 0

Player 0 wants to achieve the value $g_\gamma(v_0)$ or to come ε-close. To reach this goal, she imagines to play in $\mathcal{G}^-_\gamma$ and uses her ε-optimal strategies $\sigma^\varepsilon_\gamma$ for that game. Between every two occurrences of nodes of minimal priority throughout the play, she plays a strategy $\sigma^{\varepsilon_i}_\gamma$.



Player 0's strategy after having seen i nodes of priority m.

Initially, ε_i will be $\frac{\varepsilon}{2}$, ε being the approximation value she wants to attain in the end. Then she chooses a lower ε_{i+1} every time she passes an edge outside of $\mathcal{G}^-$. She will adjust the approximation value not only by cutting it in half every time she changes the strategy, but also according to the discount factors seen so far, since they also can dramatically alter the value of the approximation.

For a history h or a full play π, let $L(h)$ (resp. $L(\pi)$) be the number of nodes with minimal priority m occurring in h (or π).

Definition 4.10. The strategy σ^ε for Player 0 in the game $\mathcal{G}$, after history $h = v_0 \dots v_\ell$ is given as follows. In the case that $L(h) = 0$ (i.e., no position of minimal priority has been seen), let $\varepsilon' := \varepsilon/2$, and $\sigma^\varepsilon(h) := \sigma_\gamma^{\varepsilon'}(h)$. Otherwise, let v_k be the last node of priority m in the history $h = v_0 \dots v_\ell$,

$$\varepsilon' := \frac{\varepsilon}{2^{L(h)+1} D(v_0 \dots v_k)}.$$

and

$$\sigma^\varepsilon(h) := \sigma_\gamma^{\varepsilon'}(v_{k+1} \dots v_\ell).$$

Now let us consider a play $\pi = v_0 \dots v_k v_{k+1} \dots$, consistent with a strategy σ^ε, where v_k is the first node with minimal priority. The following property about values $g_\gamma(v_0)$ and $g_\gamma(v_{k+1})$ in such case (and an analogous, but more tedious one for Player 1) is the main technical point in proving ε-optimality.

Lemma 4.11. *$\Delta(v_0 \dots v_k) \cdot g_\gamma(v_{k+1})$ is $\frac{\varepsilon}{2}$-above $g_\gamma(v_0)$.*

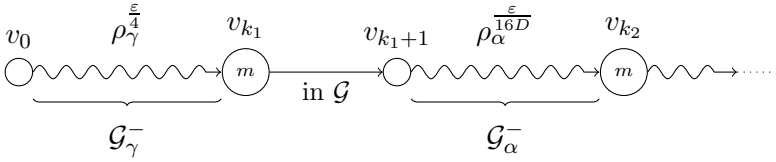
With the above lemma we prove the ε-optimality of the strategies σ^ε, as stated in the proposition below.

Proposition 4.12. *The strategy σ^ε is ε-optimal, i.e. for every $v \in V$ and every strategy ρ for Player 1, $p(\pi_{\sigma^\varepsilon,\rho}(v))$ is ε-above $g(v)$.*

The Strategy of Player 1

Now we look at the situation of Player 1. The problem of Player 1 is that he cannot just combine his strategies for $\mathcal{G}_\gamma^-$. If he did so, he would risk going infinitely often through nodes with minimal priority which is his worst case scenario. Intuitively speaking, he needs a way to count down, so that will be able to come close enough to his desired value, but will stop going through the nodes with minimal priority after a finite number of times. To achieve that, he utilises the strategy index as a counter. Like Player 0, he starts with a strategy for $\mathcal{G}_\gamma^-$, but with every strategy change at the nodes of minimal priority he not only adjusts the approximation value according to the previous one and the discount factors seen so far, but also lowers the strategy index in the following way. If the current game index is a successor ordinal, he just changes the index to its predecessor and adjusts the approximation value in the same way Player 0 does. If the current game index is a limit value, he uses the fact, that there is a game index belonging to a game which has an outcome close enough to still

reach his desired outcome. In the situation depicted below he would choose an α such that $\mathrm{val}\mathcal{G}_\alpha^-(v_{k_1+1})$ is $\frac{\varepsilon}{4}$-below $\lambda_\gamma(v_{k_1})$.



Player 1's strategy at the beginning of the play for a limit ordinal γ.

Finally, after a finite number of changes, as the ordinals are well-founded, he will be playing some version of $\rho_0^{\varepsilon_l}$ and keep on playing this strategy for the rest of the play.

Now we formally describe Player 1's strategy. Let us first fix some notation considering game indices. For a limit ordinal α, a node $v \in V$ of priority m, and for $\varepsilon \in (0,1)$, we denote by $\alpha \restriction \varepsilon, v$ the index for which the value $\mathrm{val}\mathcal{G}_\alpha^-(v)$ is ε-below $\lambda_\alpha(w)$, where $\{w\} = vE$.

Definition 4.13. For a given approximation value ε', a starting ordinal ζ, and a history $h = v_0 \dots v_l$, we define game indices $\alpha_\zeta(h, \varepsilon')$, approximation values $\varepsilon(h, \varepsilon')$, and a strategy $\rho^{\varepsilon'}$ for Player 1 in the following way.

If $L(h) = 0$, we fix $\alpha_\zeta(h, \varepsilon') = \zeta$ and $\varepsilon(h, \varepsilon') = \varepsilon'$.

For $h = v_0 \dots v_k v_{k+1} \dots v_l$, where v_k is the last node with minimal priority in h, let $h' = v_0 \dots v_{k-1}$ and put

$$\alpha_\zeta(h, \varepsilon') = \begin{cases} \alpha_\zeta(h', \varepsilon') - 1 & \text{for } \alpha_\zeta(h', \varepsilon') \text{ successor ordinal,} \\ \alpha_\zeta(h', \varepsilon') \restriction (\frac{\varepsilon'}{4^{L(h')+1} D(h')}, v_k) & \text{for } \alpha_\zeta(h', \varepsilon') \text{ limit ordinal,} \\ 0 & \text{for } \alpha_\zeta(h', \varepsilon') = 0, \end{cases}$$

and $\varepsilon(h, \varepsilon') = \frac{\varepsilon'}{4^{L(h)} D(v_0 \dots v_k)}$.

The ε'-optimal strategy for Player 1 is given by:

$$\rho_\zeta^{\varepsilon'}(v_0 \dots v_l) = \rho_{\alpha_\zeta(v_0 \dots v_l, \varepsilon')}^{\frac{\varepsilon(v_0 \dots v_l, \varepsilon')}{4}}.$$

Proposition 4.14. *The strategy ρ_ζ^ε is ε-optimal, i.e. for every $\varepsilon \in (0,1)$, for all $v \in V$, and strategies σ of Player 0: $p(\pi_{\sigma, \rho_\zeta^\varepsilon}(v))$ is ε-below $g_\zeta(v)$.*

Having defined the ε-optimal strategies σ^ε and ρ_γ^ε, we can formulate the conclusion.

Proposition 4.15. *For a QPG $\mathcal{G} = (V, E, \lambda, \Omega)$, for all $v \in V$,*

$$\sup_{\sigma \in \Gamma_0} \inf_{\rho \in \Gamma_1} p(\pi_{\sigma,\rho}(v)) = \inf_{\rho \in \Gamma_1} \sup_{\sigma \in \Gamma_0} p(\pi_{\sigma,\rho}(v)) = \mathrm{val}\mathcal{G}(v) = g(v).$$

4.3. Quantitative μ-calculus and Games

After establishing determinacy for quantitative parity games we are ready to prove Theorem 4.4. In the proof, we first use structural induction to show that $\mathrm{MC}[\mathcal{K}, \varphi]$ is a model checking game for QML formulae. Further, we only need to inductively consider formulae of the form $\varphi = \nu X.\psi$.

Note that in the game $\mathrm{MC}[\mathcal{Q}, \varphi]$, the positions with minimal priority are of the form (X, v) each with a unique successor (φ, v). Our induction hypothesis states that for every interpretation g of the fixed-point variable X, it holds that:

$$[\![\varphi]\!]^{\mathcal{Q}}_{[X \leftarrow g]} = \mathrm{valMC}[\mathcal{Q}, \psi[X/g]]. \tag{4.1}$$

By Theorem 2.4, we know that we can compute $\nu X.\psi$ inductively in the following way: $[\![\nu X.\psi]\!]^{\mathcal{K}}_{\varepsilon} = g_\gamma$ with $g_0(v) = \infty$ for all $v \in V$ and

$$g_\alpha = \begin{cases} [\![\psi]\!]_{\varepsilon[X \leftarrow g_{\alpha-1}]} & \text{for } \alpha \text{ successor ordinal,} \\ \lim_{\beta<\alpha} [\![\psi]\!]_{\varepsilon[X \leftarrow g_\beta]} & \text{for } \alpha \text{ limit ordinal,} \end{cases}$$

and where $g_\gamma = g_{\gamma+1}$.

Now we want to prove that the games $\mathrm{MC}[\mathcal{Q}, \psi[X/g_\alpha]]$ coincide with the unfolding of $\mathrm{MC}[\mathcal{Q}, \varphi]$. We say that two games coincide if the game graph is essentially the same, except for some additional moves where neither player has an actual choice and there is no discount that could change the outcome. In our case these are the moves from $\varphi = \nu X.\psi$ to ψ, which allows us to show the following lemma.

Lemma 4.16. *The games* $\mathrm{MC}[\mathcal{Q}, \psi[X/g_\alpha]]$ *and* $\mathrm{MC}[\mathcal{Q}, \varphi]^-_\alpha$ *coincide for all* α.

From the above and Proposition 4.15, we conclude that the value of the game $\mathrm{MC}[\mathcal{Q}, \varphi]$ is the limit of the values $\mathrm{MC}[\mathcal{Q}, \varphi]^-_\alpha$, whose value functions coincide with the stages of the fixed-point evaluation g_α for all α, and thus

$$\mathrm{valMC}[\mathcal{Q}, \varphi] = \mathrm{valMC}[\mathcal{Q}, \varphi]^-_\gamma = g_\gamma = [\![\varphi]\!]^{\mathcal{Q}}.$$

■

5. Describing Game Values in $\mathrm{Q}\mu$

Having model checking games for the quantitative μ-calculus is just one direction in the relation between games and logic. The other direction concerns the definability of the winning regions in a game by formulae in the corresponding logic. For the classical μ-calculus such formulae have been constructed by Walukiewicz and it has been shown that for any parity game of fixed priority they define the winning region for Player 0, see e.g. [9]. We extend this theorem to the quantitative case in the following way. We represent quantitative parity games $(V, V_0, V_1, E, \delta_G, \lambda_G, \Omega_G)$ with priorities $\Omega(V) \in \{0, \dots d-1\}$ by a quantitative transition system $\mathcal{Q}_\mathcal{G} = (V, E, \delta, V_0, V_1, \Lambda, \Omega)$, where $V_i(v) = \infty$ when $v \in V_i$ and $V_i(v) = 0$ otherwise, $\Omega(v) = \Omega_G(v)$ when $vE \neq \emptyset$ and $\Omega(v) = d$ otherwise,

$$\delta(v, w) = \begin{cases} \delta_G(v, w) & \text{when } v \in V_0, \\ \frac{1}{\delta_G(v,w)} & \text{when } v \in V_1, \end{cases}$$

and payoff predicate $\Lambda(v) = \lambda_G(v)$ when $vE = \emptyset$ and $\Lambda(v) = 0$ otherwise.

We then build the formula Win_d and formulate the theorem

$$\mathrm{Win}_d = \nu X_0.\mu X_1.\nu X_2.\dots.\lambda X_{d-1} \bigvee_{j=0}^{d-1} ((V_0 \wedge P_j \wedge \Diamond X_j) \vee (V_1 \wedge P_j \wedge \Box X_j)) \vee \Lambda,$$

where $\lambda = \nu$ if d is odd, and $\lambda = \mu$ otherwise, and $P_i := \neg(\mu X.(2 \cdot X \vee |\Omega - i|))$.

Theorem 5.1. *For every* $d \in \mathbb{N}$, *the value of any quantitative parity game* $\mathcal{G}$ *with priorities in* $\{0, \dots d-1\}$ *coincides with the value of* Win_d *on the associated transition system* $\mathcal{Q}_\mathcal{G}$.

6. Conclusions and Future Work

In this work, we showed how the close connection between the modal μ-calculus and parity games can be lifted to the quantitative setting, provided that the quantitative extensions of the logic and the games are defined in an appropriate manner. This is just a first step in a systematic investigation of what connections between logic and games survive in the quantitative setting. These investigations should as well be extended to quantitative variants of other logics, in particular LTL, CTL, CTL*, and PDL.

Following [3] we work with games where discounts are multiplied along edges and values range over the non-negative reals with infinity. Another natural possibility is to use addition instead of multiplication and let the values range over the reals with $-\infty$ and $+\infty$. Crash games, recently introduced in [7], are defined in such a way, but with values restricted to integers. Gawlitza and Seidl present an algorithm for crash games over finite graphs which is based on strategy improvement [7]. It is possible to translate back and forth between quantitative parity games and crash games with real values by taking logarithms of the discount values on edges as payoffs for moves in the crash game. The exponent of the value of such a crash game is then equal to the value of the original quantitative parity game. This suggests that the methods from [7] can be applied to quantitative parity games as well. This could lead to efficient model-checking algorithms for Qμ and would thus further justify the game-based approach to model checking modal logics.

References

[1] Luca de Alfaro. Quantitative verification and control via the mu-calculus. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR*, volume 2761 of *LNCS*, pages 102–126. Springer, 2003.

[2] Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. Model checking discounted temporal properties. *Theoretical Computer Science*, 345(1):139–170, 2005.

[3] Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. Technical Report ucsc-crl-05-01, School of Engineering, University of California, Santa Cruz, 2005.

[4] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 1022–1037. Springer, 2003.

[5] Luca de Alfaro and Rupak Majumdar. Quantitative solution of omega-regular games. *J. Comput. Syst. Sci.*, 68(2):374–397, 2004.

[6] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model-checking for fragments of μ-calculus. In *CAV 93*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396. Springer, 1993.

[7] Thomas Gawlitza and Helmut Seidl. Computing game values for crash games. In Kedar S. Namjoshi *et al.*, eds, *ATVA*, *Lect. Notes in Comp. Science* 4762, pp. 177-191. Springer, 2007.

[8] Hugo Gimbert and Wieslaw Zielonka. Perfect information stochastic priority games. In Lars Arge *et al.*, eds, *ICALP*, *Lect. Notes in Comp. Science* 4596, pp. 850-861. Springer, 2007.

[9] Erich Grädel. Finite model theory and descriptive complexity. In *Finite Model Theory and Its Applications*, pages 125–230. Springer-Verlag, 2007.

[10] Marcin Jurdziński. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000.

[11] Annabelle McIver and Carroll Morgan. Results on the quantitative μ-calculus qMμ. *ACM Trans. Comput. Log.*, 8(1), 2007.

[12] Colin Stirling. Games and modal mu-calculus. In Tiziana Margaria and Bernhard Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 1996.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 313-324
www.stacs-conf.org

ORDER-INVARIANT MSO IS STRONGER THAN COUNTING MSO IN THE FINITE

TOBIAS GANZOW [1] AND SASHA RUBIN [2]

[1] Mathematische Grundlagen der Informatik, RWTH Aachen, Germany
E-mail address: ganzow@logic.rwth-aachen.de

[2] Department of Computer Science, University of Auckland, New Zealand
E-mail address: rubin@cs.auckland.ac.nz

Abstract. We compare the expressiveness of two extensions of monadic second-order logic (MSO) over the class of finite structures. The first, counting monadic second-order logic (CMSO), extends MSO with first-order modulo-counting quantifiers, allowing the expression of queries like "the number of elements in the structure is even". The second extension allows the use of an additional binary predicate, not contained in the signature of the queried structure, that must be interpreted as an arbitrary linear order on its universe, obtaining order-invariant MSO.

While it is straightforward that every CMSO formula can be translated into an equivalent order-invariant MSO formula, the converse had not yet been settled. Courcelle showed that for restricted classes of structures both order-invariant MSO and CMSO are equally expressive, but conjectured that, in general, order-invariant MSO is stronger than CMSO.

We affirm this conjecture by presenting a class of structures that is order-invariantly definable in MSO but not definable in CMSO.

1. Introduction

Linear orders play an important role in descriptive complexity theory since certain results relating the expressive power of logics to complexity classes, e.g., the Immerman-Vardi Theorem that LFP captures Ptime, only hold for classes of linearly ordered structures. Usually, the order only serves to systematically access all elements of the structure, and consequently to encode the configurations of a step-wise advancing computation of a Turing machine by tuples of elements of the structure. In these situations we do not actually want to make statements about the properties of the order, but merely want to have an arbitrary linear order available to express the respective coding techniques.

Furthermore, when actually working with finite structures in an algorithmic context, e.g., when evaluating queries in a relational database, we are in fact working on an implicitly ordered structure since, although relations in a database are modelled as *sets* of tuples, the relations are nevertheless stored as *ordered sequences* of tuples in memory or on a disk. As

1998 ACM Subject Classification: F.4.1 Mathematical Logic.
Key words and phrases: MSO, Counting MSO, order-invariance, expressiveness, Ehrenfeucht-Fraïssé game.

this linear order is always available (though, as in the case of databases, it is implementation-dependent and may even change over time as tuples are inserted or deleted), we could allow queries to make use of an additional binary predicate that is interpreted as a linear order on the universe of the structure, but require the outcome of the query not to depend on the actual ordering, but to be *order-invariant*. Precisely, given a τ-structure $\mathfrak{A}$, we allow queries built over an expanded vocabulary $\tau \dot\cup \{<\}$, and say that a query φ is *order-invariant* if $(\mathfrak{A}, <_1) \models \varphi \iff (\mathfrak{A}, <_2) \models \varphi$ for all possible relations $<_1$ and $<_2$ linearly ordering A.

Using Ehrenfeucht-Fraïssé-games for MSO, one can see that MSO on sets (i.e., structures over an empty vocabulary) is too weak to express that the universe contains an even number of elements. However, this is possible if the universe is linearly ordered: simply use the MSO sentence stating that the maximal element should be contained in the set of elements on even positions in the ordering. Obviously, such a sentence is order-invariant since rearranging the elements does not affect its truth value. Gurevich uses this observation to show that the property of Boolean algebras having an even number of atoms, although not definable in FO, is order-invariantly definable in FO (simulating the necessary MSO-quantification over sets of atoms by FO-quantification over the elements of the Boolean algebra).

If we explicitly add modulo-counting to MSO, e.g., via modulo-counting first-order quantifiers such as "there exists an even number of elements x such that ...", we obtain *counting monadic second-order logic* (CMSO), and the question naturally arises as to whether there are properties not expressible in CMSO that can be expressed order-invariantly in MSO.

In fact, a second separation example due to Otto gives a hint in that direction. The class of structures presented in [Ott00] even separates order-invariant FO from FO extended by arbitrary unary generalised quantifiers, i.e., especially modulo-counting quantifiers, and exploits the idea of "hiding" a part of the structure such that it is only meaningfully usable for queries in presence of a linear order (or, as actually proven in the paper, in presence of an arbitrary choice function).

The expressiveness of CMSO has been studied, e.g., in [Cou90], where it is mainly compared to MSO, and in [Cou96] it is shown that, on the class of forests, order-invariant MSO is no more expressive than CMSO. As pointed out in [BS05], this can be generalised using results in [Lap98] to classes of structures of bounded tree-width. But still, this left open Courcelle's conjecture: that order-invariant MSO is strictly stronger than CMSO for general graphs [Cou96, Conjecture 7.3].

In this paper, we present a suitable characterisation of CMSO-definability in terms of an Ehrenfeucht-Fraïssé game, and later, as the main contribution, we present a separating example showing that a special class of graphs is indeed definable by an order-invariant MSO sentence but not by a counting MSO sentence.

2. Preliminaries

Throughout the paper $\mathbb{N}$ denotes the set of non-negative integers and $\mathbb{N}^+ := \mathbb{N} - \{0\}$. Given a non-empty finite set $M = \{m_1, \dots, m_k\} \subseteq_{\text{fin}} \mathbb{N}^+$, let $\operatorname{lcm}(M) := \operatorname{lcm}(m_1, \dots, m_k)$ denote the least common multiple of all elements in M; additionally, we define $\operatorname{lcm}(\emptyset) = 1$. For sets X and Y as well as M as before, we abbreviate that $|X| \equiv |Y| \pmod m$ for all $m \in M$ by using the shorthand $|X| \equiv |Y| \pmod M$.

We restrict our attention to finite τ-structures with a nonempty universe over a countable relational vocabulary τ, possibly with constants, and we will mainly deal with monadic second-order logic and some of its extensions. For more details concerning finite model theory, we refer to [EF95] or [Lib04].

When comparing the expressiveness of two logics $\mathcal{L}$ and $\mathcal{L}'$, we say that $\mathcal{L}'$ *is at least as expressive as* $\mathcal{L}$, denoted $\mathcal{L} \subseteq \mathcal{L}'$, if for every $\varphi \in \mathcal{L}[\tau]$ there exists a $\varphi' \in \mathcal{L}'[\tau]$ such that $\mathrm{Mod}(\varphi) = \mathrm{Mod}(\varphi')$, where $\mathrm{Mod}(\varphi)$ denotes the class of all finite τ-structures satisfying φ.

2.1. Counting MSO

The notion of (modulo-)counting monadic second-order logic (CMSO) can be introduced in two different, but nonetheless equivalent, ways. The first view of CMSO is via an extension of MSO by modulo-counting first-order quantifiers.

Definition 2.1. Let τ be a signature and $M \subseteq \mathbb{N}^+$ a set of moduli, then

- every formula $\varphi \in \mathrm{MSO}[\tau]$ is also a formula in $\mathrm{CMSO}^{(M)}[\tau]$, and
- if $\varphi(x) \in \mathrm{CMSO}^{(M)}[\tau]$ and $m \in M$, then $\exists^{(m)}x.\varphi(x) \in \mathrm{CMSO}^{(M)}[\tau]$.

If we do not restrict the set of modulo-counting quantifiers being used, we get the full language $\mathrm{CMSO}[\tau] = \mathrm{CMSO}^{(\mathbb{N}^+)}[\tau]$. The semantics of MSO formulae is as expected, and we have $\mathfrak{A} \models \exists^{(m)}x.\varphi(x)$ if and only if $|\{a \in A : \mathfrak{A} \models \varphi(a)\}| \equiv 0 \pmod m$. The quantifier rank $\mathrm{qr}(\psi)$ of a $\mathrm{CMSO}[\tau]$ formula ψ is defined as for MSO-formulae with the additional rule that $\mathrm{qr}\left(\exists^{(m)}x.\varphi(x)\right) = 1 + \mathrm{qr}(\varphi)$, i.e., we do not distinguish between different kinds of quantifiers.

In this paper we use an alternative but equivalent definition of CMSO, namely the extension of the MSO language by monadic second-order predicates $C^{(m)}$ which hold true of a set X if and only if $|X| \equiv 0 \pmod m$. As in the definition above, formulae of the fragment $\mathrm{CMSO}^{(M)}[\tau]$ may only use predicates $C^{(m)}$ where $m \in M$. The back-and-forth translation can be carried out along the following equivalences which increase the quantifier rank by at most one in each step:

$$\begin{aligned} \exists^{(m)}x.\varphi(x) &\equiv \exists X(C^{(m)}(X) \wedge \forall x(Xx \leftrightarrow \varphi(x))) \quad \text{and} \\ C^{(m)}(X) &\equiv \exists^{(m)}x.Xx\,. \end{aligned}$$

Furthermore, the introduction of additional predicates $C^{(m,r)}$ (or, equivalently, additional modulo-counting quantifiers $\exists^{(m,r)}$) stating for a set X that $|X| \equiv r \pmod m$ does not increase the expressive power since they can be simulated as follows (with only a constant increase of quantifier rank):

$$C^{(m,r)}(X) \equiv \exists X_0(\text{“}X_0 \subseteq X\text{”} \wedge \text{“}|X_0| = r\text{”} \wedge \text{“}C^{(m)}(X \setminus X_0)\text{”})\,,$$

where all subformulae are easily expressible in MSO.

Later, we will introduce an Ehrenfeucht-Fraïssé game capturing the expressiveness of CMSO with this extended set of second-order predicates.

2.2. Order-invariance

Let τ be a relational vocabulary and $\varphi \in \mathrm{MSO}[\tau \dot\cup \{<\}]$, i.e., φ may contain an additional relation symbol $<$. Then φ is called *order-invariant on a class $\mathcal{C}$ of τ-structures* if, and only if, $(\mathfrak{A}, <_1) \models \varphi \iff (\mathfrak{A}, <_2) \models \varphi$ for all $\mathfrak{A} \in \mathcal{C}$ and all linear orders $<_1$ and $<_2$ on A.

Although, in general, it is undecidable whether a given MSO-formula is order-invariant in the finite, we will speak of the *order-invariant fragment of* MSO, denoted by $\mathrm{MSO}[<]_{inv}$, that contains all formulae that are order-invariant on the class of all finite structures.

It is an easy observation that every CMSO formula is equivalent over the class of all finite structures to an order-invariant MSO formula by translating counting quantifiers in the following way:

$$\exists^{(q)} x.\varphi(x) \;:=\; \exists X \exists X_0 \dots \exists X_{q-1} \left(\begin{array}{l} \forall x \, (Xx \leftrightarrow \varphi(x)) \wedge \;\text{“}\{X_0, \dots, X_{q-1}\} \text{ is a partition of } X\text{”} \\ \wedge\; \exists x \big(X_0 x \wedge \forall y (Xy \to x \leq y)\big) \wedge\; \exists x \big(X_{q-1} x \wedge \forall y (Xy \to x \geq y)\big) \\ \wedge\; \forall x \forall y \left(S_{\varphi,<}(x,y) \to \left(\bigwedge_{i=0}^{q-1} X_i x \leftrightarrow X_{i+1 \pmod q} y \right) \right) \end{array} \right)$$

where $S_{\varphi,<}$ defines the successor relation induced by an arbitrary order $<$ on the universe of the structure restricted to the set X of elements for which φ holds.

Note that the quantifier rank of the translated formula is not constant but bounded by the parameter in the counting quantifier.

3. An Ehrenfeucht-Fraïssé game for CMSO

The Ehrenfeucht-Fraïssé game capturing expressiveness of MSO parameterised by the quantifier-rank (cf. [EF95, Lib04]) can be naturally extended to a game capturing the expressiveness of CMSO parameterised by the quantifier rank and the set of moduli being used in the cardinality predicates or counting quantifiers.

Viewing CMSO as MSO with additional quantifiers $\exists^{(m)} x.\varphi(x)$ for all m in a fixed set M leads to a new type of move described, e.g., in the context of extending FO by modulo-counting quantifiers in [Nur00]. Since a modulo-counting quantifier actually combines notions of a first-order and a monadic second-order quantifier in the sense that it makes a statement about the cardinality of a certain *set* of elements, but on the other hand, it behaves like a first-order quantifier binding an *element* variable and making a statement about that particular element, the move capturing modulo-counting quantification consists of two phases. First, Spoiler and Duplicator select sets of elements S and D in the structures such that $|S| \equiv |D| \pmod M$, and in the second phase, Spoiler and Duplicator select elements a and b such that $a \in S$ if and only if $b \in D$. After the move, reflecting the first-order nature of the quantifier, only the two selected elements a and b are remembered and contribute to the next position in the game, whereas the information about the chosen sets is discarded.

We prefer viewing CMSO via second-order cardinality predicates, yielding an Ehrenfeucht-Fraïssé game that allows a much clearer description of winning strategies. Since we do not have additional quantifiers, we have exactly the same types of moves as in the Ehrenfeucht-Fraïssé game for MSO, and we merely modify the winning condition to take the new predicates into account.

Towards this end, we first introduce a suitable concept of partial isomorphisms between structures.

Definition 3.1. With any structure $\mathfrak{A}$ and any set $M \subseteq_{\text{fin}} \mathbb{N}^+$ we associate the (first-order) power set structure $\mathfrak{A}^M := \big(\mathcal{P}(A), (C^{(m,r)})_{\substack{m\in M\\ 0\le r<m}}\big)$, where the predicates $C^{(m,r)}$ are interpreted in the obvious way. (Note that first-order predicates in the power set structure $\mathfrak{A}^M$ naturally correspond to second-order predicates in $\mathfrak{A}$.)

Let $\mathfrak{A}$ and $\mathfrak{B}$ be τ-structures, and let $M \subseteq_{\text{fin}} \mathbb{N}^+$ be a fixed set of moduli. Then the mapping $(A_1, \dots, A_s, a_1, \dots, a_t) \mapsto (B_1, \dots, B_s, b_1, \dots, b_t)$ is called a *twofold partial isomorphism between $\mathfrak{A}$ and $\mathfrak{B}$ with respect to M* if

(i) $(a_1, \dots, a_t) \mapsto (b_1, \dots, b_t)$ is a partial isomorphism between $(\mathfrak{A}, A_1, \dots, A_s)$ and $(\mathfrak{B}, B_1, \dots, B_s)$ and
(ii) $(A_1, \dots, A_s) \mapsto (B_1, \dots, B_s)$ is a partial isomorphism between $\mathfrak{A}^M$ and $\mathfrak{B}^M$.

We propose the following Ehrenfeucht-Fraïssé game to capture the expressiveness of CMSO where the use of moduli is restricted to a (finite) set M and formulae of quantifier rank at most r.

Definition 3.2 (Ehrenfeucht-Fraïssé game for CMSO)**.** Let $M \subseteq_{\text{fin}} \mathbb{N}^+$ and $r \in \mathbb{N}$. The r-round (mod M) Ehrenfeucht-Fraïssé game $\mathcal{G}_r^M(\mathfrak{A}, \mathfrak{B})$ is played by Spoiler and Duplicator on τ-structures $\mathfrak{A}$ and $\mathfrak{B}$. In each turn, Spoiler can choose between the following types of moves:

- *point move:* Spoiler selects an element in one of the structures, and Duplicator answers by selecting an element in the other structure.
- *set move:* Spoiler selects a set of elements X in one of the structures, and Duplicator responds by choosing a set of elements Y in the other structure.

After $r = s + t$ rounds, when the players have chosen sets $A_1, \dots, A_s$ and $B_1, \dots, B_s$ as well as elements $a_1, \dots, a_t$ and $b_1, \dots, b_t$ in an arbitrary order, Duplicator wins the game if, and only if, $(A_1, \dots, A_s, a_1, \dots, a_t) \mapsto (B_1, \dots, B_s, b_1, \dots, b_t)$ is a twofold partial isomorphism between $\mathfrak{A}$ and $\mathfrak{B}$ with respect to M.

First note that, although Duplicator is required to answer a set move X by a set Y such that $|X| \equiv |Y| \pmod M$ in order to win, we do not have to make this explicit in the rules of the moves since these cardinality constraints are already imposed by the winning condition (X and Y would not define a twofold partial isomorphism if they did not satisfy the same cardinality predicates). Furthermore, for $M = \emptyset$ or $M = \{1\}$, the resulting game $\mathcal{G}_r^M(\mathfrak{A}, \mathfrak{B})$ corresponds exactly to the usual Ehrenfeucht-Fraïssé game for MSO.

Theorem 3.3. *Let $\mathfrak{A}$ and $\mathfrak{B}$ be τ-structures, $r \in \mathbb{N}$, and $M \subseteq_{\text{fin}} \mathbb{N}$. Then the following are equivalent:*

(i) $\mathfrak{A} \equiv_r^M \mathfrak{B}$, i.e., $\mathfrak{A} \models \varphi$ if and only if $\mathfrak{B} \models \varphi$ for all $\varphi \in \mathrm{CMSO}^{(M)}[\tau]$ with $\mathrm{qr}(\varphi) \le r$.
(ii) Duplicator has a winning strategy in the r-round (mod M) Ehrenfeucht-Fraïssé game $\mathcal{G}_r^M(\mathfrak{A}, \mathfrak{B})$. ∎

To prove non-definability results, we can make use of the following standard argument.

Proposition 3.4. *A class $\mathcal{C}$ of τ-structures is not definable in CMSO if, for every $r \in \mathbb{N}$ and every $M \subseteq_{\text{fin}} \mathbb{N}^+$, there are τ-structures $\mathfrak{A}_{M,r}$ and $\mathfrak{B}_{M,r}$ such that $\mathfrak{A}_{M,r} \in \mathcal{C}$, $\mathfrak{B}_{M,r} \notin \mathcal{C}$, and $\mathfrak{A}_{M,r} \equiv_r^M \mathfrak{B}_{M,r}$.*

The following lemma, stating that the CMSO-theory of disjoint unions can be deduced from the CMSO-theories of the components, can either be proved, as carried out in [Cou90, Lemma 4.5], by giving an effective translation of sentences talking about the disjoint union of two structures into a Boolean combination of sentences each talking about the individual structures, or by using a game-oriented view showing that winning strategies for Duplicator in the games on two pairs of structures can be combined into a winning strategy on the pair of disjoint unions of the structures.

Lemma 3.5. *Let* $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{B}_1$, *and* $\mathfrak{B}_2$ *be* τ*-structures such that* $\mathfrak{A}_1 \equiv_r^M \mathfrak{B}_1$ *and* $\mathfrak{A}_2 \equiv_r^M \mathfrak{B}_2$. *Then* $\mathfrak{A}_1 \dot\cup \mathfrak{A}_2 \equiv_r^M \mathfrak{B}_1 \dot\cup \mathfrak{B}_2$.

Proof. Consider the game on $\mathfrak{A} := \mathfrak{A}_1 \dot\cup \mathfrak{A}_2$ and $\mathfrak{B} := \mathfrak{B}_1 \dot\cup \mathfrak{B}_2$. A Spoiler's point move in $\mathfrak{A}$ (resp., in $\mathfrak{B}$) is answered by Duplicator according to her winning strategy in either $\mathcal{G}_r^M(\mathfrak{A}_1, \mathfrak{B}_1)$ or $\mathcal{G}_r^M(\mathfrak{A}_2, \mathfrak{B}_2)$. A set move $S \subseteq A$ (analogous for $S \subseteq B$) is decomposed into two subsets $S_1 := S \cap A_1$ and $S_2 := S \cap A_2$, and is answered by Duplicator by the set $D := D_1 \cup D_2$ consisting of the sets D_1 and D_2 chosen according to her winning strategies as responses to S_1 and S_2 in the respective games $\mathcal{G}_r^M(\mathfrak{A}_1, \mathfrak{B}_1)$ and $\mathcal{G}_r^M(\mathfrak{A}_2, \mathfrak{B}_2)$.

Since A_1 and A_2 as well as B_1 and B_2 are disjoint, we have $|S| = |S_1| + |S_2|$ and $|D| = |D_1| + |D_2|$. Furthermore, $|S_1| \equiv |D_1| \pmod M$ and $|S_2| \equiv |D_2| \pmod M$ as the sets D_1 and D_2 are chosen according to Duplicator's winning strategies in the games on $\mathfrak{A}_1$ and $\mathfrak{B}_1$, and $\mathfrak{A}_2$ and $\mathfrak{B}_2$, respectively. Since $\equiv \pmod M$ is a congruence relation with respect to addition, we have that $|S| \equiv |D| \pmod M$. It is easily verified that the sets and elements chosen according to this strategy indeed define a twofold partial isomorphism between $\mathfrak{A}$ and $\mathfrak{B}$. ∎

As a direct corollary we obtain the following result that will be used in the inductive step in the forthcoming proofs.

Corollary 3.6. *Let* $\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{B}_1$, *and* $\mathfrak{B}_2$ *be* τ*-structures, such that* $\mathfrak{A}_1 \equiv_r^M \mathfrak{B}_1$ *and* $\mathfrak{A}_2 \equiv_r^M \mathfrak{B}_2$. *Then* $(\mathfrak{A}_1 \dot\cup \mathfrak{A}_2, A_1) \equiv_r^M (\mathfrak{B}_1 \dot\cup \mathfrak{B}_2, B_1)$.

Proof. We consider the following $\tau \dot\cup \{P\}$-expansions of the given structures: $\mathfrak{A}_1' := (\mathfrak{A}_1, A_1)$, $\mathfrak{B}_1' := (\mathfrak{B}_1, B_1)$, $\mathfrak{A}_2' := (\mathfrak{A}_2, \emptyset)$, and $\mathfrak{B}_2' := (\mathfrak{B}_2, \emptyset)$. It is immediate that

(i) $\mathfrak{A}_1 \equiv_r^M \mathfrak{B}_1$ implies $(\mathfrak{A}_1, A_1) \equiv_r^M (\mathfrak{B}_1, B_1)$, and

(ii) $\mathfrak{A}_2 \equiv_r^M \mathfrak{B}_2$ implies $(\mathfrak{A}_2, \emptyset) \equiv_r^M (\mathfrak{B}_2, \emptyset)$

since Duplicator can obviously win the respective Ehrenfeucht-Fraïssé games on the expanded structures using the same strategies as in the games proving the equivalences on the left-hand side. The claim follows by applying the previous lemma to the $\tau \dot\cup \{P\}$-expansions. ∎

It is well known that MSO exhibits a certain weakness regarding the ability to specify cardinality constraints on sets, i.e., structures over an empty vocabulary. A proof of this fact using Ehrenfeucht-Fraïssé games can be found in [Lib04]. By adapting this proof, we show that this is still the case for CMSO.

Lemma 3.7. *Let* $\mathfrak{A}$ *and* $\mathfrak{B}$ *be* $\emptyset$*-structures,* $M \subseteq_{\mathit{fin}} \mathbb{N}^+$, *and* $r \in \mathbb{N}$. *Then* $\mathfrak{A} \equiv_r^M \mathfrak{B}$ *if* $|A|, |B| \geq (2^{r+1} - 4)\operatorname{lcm}(M)$ *and* $|A| \equiv |B| \pmod M$.

Proof. We prove by induction on the number of rounds that Duplicator wins the $(\mathrm{mod}\ M)$ r-round Ehrenfeucht-Fraïssé game $\mathcal{G}_r^M(\mathfrak{A}, \mathfrak{B})$. For $r = 0$ and $r = 1$ the claim is obviously

true. Let $r > 1$, assume that the claim holds for $r-1$, and consider the first move of the r-round game. We assume that Spoiler makes his move in $\mathfrak{A}$ since the reasoning in the other case is completely symmetric.

If Spoiler makes a set move $S \subseteq A$, we consider the following cases:

(1) $|S| < (2^r - 4) \cdot \operatorname{lcm}(M)$ (or $|A - S| < (2^r - 4) \cdot \operatorname{lcm}(M)$). Then Duplicator selects a set $D \subseteq B$ such that $|D| = |S|$ (or $|B - D| = |A - S|$), and hence $S \cong D$ and $A - S \equiv^M_{r-1} B - D$ (or $A - S \cong B - D$ and $S \equiv^M_{r-1} D$).

(2) $|S|, |A-S| \geq (2^r-4)\cdot\operatorname{lcm}(M)$. Then Duplicator selects a set $D \subseteq B$ such that $|D| \equiv |S| \pmod M$ and $|D|, |B - D| \geq (2^r - 2) \cdot \operatorname{lcm}(M)$. In fact, she chooses for D half of the elements and chooses $\ell < \operatorname{lcm}(M)$ additional ones to fulfil the cardinality constraints $|D| \equiv |S| \pmod M$. Then, for the set $B - D$ of non-selected elements, we have

$$\begin{aligned} |B - D| &\geq \frac{1}{2}\big((2^{r+1} - 4)\operatorname{lcm}(M)\big) - \ell \geq (2^r - 2)\operatorname{lcm}(M) - \operatorname{lcm}(M) \\ &\geq (2^r - 4)\operatorname{lcm}(M) \end{aligned}$$

for all ℓ satisfying $0 \leq \ell < \operatorname{lcm}(M)$. Since $|D| = |B - D| + 2\ell$, obviously $|D| \geq (2^r - 4)\operatorname{lcm}(M)$ as well.

Thus, in both cases, by the induction hypothesis we get $S \equiv^M_{r-1} D$ and $A - S \equiv^M_{r-1} B - D$. Hence, by Corollary 3.6 $(A, S) \equiv^M_{r-1} (B, D)$, i.e., Duplicator has a winning strategy in the remaining $(r-1)$-round game from position (S, D).

If Spoiler makes a point move $s \in A$, Duplicator answers by choosing an arbitrary element $d \in B$. Similar to Case 1 above, we observe that $(\{s\}, s) \cong (\{d\}, d)$ and $A - \{s\} \equiv^M_{r-1} B - \{d\}$ by the induction hypothesis. Thus, by Lemma 3.5, $(A, s) \equiv^M_{r-1} (B, d)$ implying that Duplicator has a winning strategy for the remaining $r-1$ rounds from position (s, d). ∎

4. The Separating Example

We will first give a brief description of our example showing that $\mathrm{MSO}[<]_{inv}$ is strictly more expressive than CMSO. We consider a property of two-dimensional grids, namely that the vertical dimension divides the horizontal dimension. This property is easily definable in MSO for grids that are given as directed graphs with two edge relations, one for the horizontal edges pointing rightwards, and one for the vertical edges pointing upwards, by defining a new relation of diagonal edges combining one step rightwards and one step upwards wrapping around from the top border to the bottom border but not from the right to the left border. Note that there is a path following those diagonal edges starting from the bottom-left corner of the grid and ending in the top-right corner if, and only if, the vertical dimension divides the horizontal dimension of the grid. Thus, for our purposes, we have to weaken the structure in the sense that we hide information that remains accessible to $\mathrm{MSO}[<]_{inv}$-formulae but not to CMSO formulae.

An appropriate loss of information is achieved by replacing the two edge relations with their reflexive symmetric transitive closure, i.e., we consider grids as structures with two equivalence relations which provide a notion of *rows* and *columns* of the grid. Obviously, notions like corner and border vertices as well as the notion of an order on the rows and columns that were important for the MSO-definition of the divisibility property are lost, but clearly, all these notions can be regained in presence of an order. First, the order allows us to uniquely define an element (e.g. the $<$-least element) to be the bottom-left corner of

the grid, and second, the order induces successor relations on the set of columns and the set of rows, from which both horizontal and vertical successor vertices of any vertex can be deduced. Since the divisibility property is obviously invariant with respect to the ordering of the rows or columns, this allows for expressing it in $\mathrm{MSO}[<]_{inv}$. In the course of this section we will develop the arguments showing that CMSO fails to express this property on the following class of grid-like structures.

Definition 4.1. A *cliquey* (k,ℓ)*-grid* is a $\{\sim_h,\sim_v\}$-structure that is isomorphic to $\mathfrak{G}_{k\ell} := (\{0,\dots,k-1\}\times\{0,\dots,\ell-1\},\sim_h,\sim_v)$, where

$$\begin{aligned}\sim_h &:= \{((x,y),(x',y')) : x = x'\} \text{ and}\\ \sim_v &:= \{((x,y),(x',y')) : y = y'\},\end{aligned}$$

i.e., $\sim_h$ consists of exactly k equivalence classes (called *rows*), each containing ℓ elements, and $\sim_v$ consists of exactly ℓ equivalence classes (called *columns*), each containing k elements, such that every equivalence class of $\sim_h$ intersects every equivalence class of $\sim_v$ in exactly one element and vice versa.

A *horizontally coloured cliquey* (k,ℓ)*-grid*, denoted $\mathfrak{G}^{\mathrm{col}}_{k\ell}$, is the expansion of the $\{\sim_v\}$-reduct of the cliquey grid $\mathfrak{G}_{k\ell}$ by unary predicates $\{P_1,\dots,P_k\}$, where the information of $\sim_h$ is retained in the k new predicates (in the following referred to as *colours*) such that each set P_i corresponds to exactly one former equivalence class.

Note that the same class of grid-like structures has already been used by Otto in a proof showing that the number of monadic second-order quantifiers gives rise to a strict hierarchy over finite structures [Ott95].

The class is first-order definable by a sentence ψ_{grid} stating that

- $\sim_v$ and $\sim_h$ are equivalence relations, and
- every pair consisting of one equivalence class of $\sim_h$ and $\sim_v$ each has exactly one element in common

as these properties are sufficient to enforce the desired grid-like structure. Note that even the second property is first-order definable since every equivalence class is uniquely determined by each of its elements.

The following two lemmata justify the introduction of the notion of horizontally coloured cliquey grids for use in the forthcoming proofs.

Lemma 4.2. *Let* $\mathfrak{G}^{\mathrm{col}}_{k\ell_1}$, $\mathfrak{G}^{\mathrm{col}}_{k\ell_2}$, $\mathfrak{G}^{\mathrm{col}}_{k\ell'_1}$, *and* $\mathfrak{G}^{\mathrm{col}}_{k\ell'_2}$ *be horizontally coloured cliquey grids such that* $\mathfrak{G}^{\mathrm{col}}_{k\ell_1} \equiv^M_r \mathfrak{G}^{\mathrm{col}}_{k\ell'_1}$ *and* $\mathfrak{G}^{\mathrm{col}}_{k\ell_2} \equiv^M_r \mathfrak{G}^{\mathrm{col}}_{k\ell'_2}$. *Then* $\mathfrak{G}^{\mathrm{col}}_{k,\ell_1+\ell_2} \equiv^M_r \mathfrak{G}^{\mathrm{col}}_{k,\ell'_1+\ell'_2}$.

Proof. Note that, since there are no horizontal edges in horizontally coloured cliquey grids and the vertical dimension of all grids is k, $\mathfrak{G}^{\mathrm{col}}_{k,\ell_1+\ell_2}$ is the disjoint union of the two smaller horizontally coloured cliquey grids $\mathfrak{G}^{\mathrm{col}}_{k\ell_1}$ and $\mathfrak{G}^{\mathrm{col}}_{k\ell_2}$, and of course, the same holds for $\mathfrak{G}^{\mathrm{col}}_{k,\ell'_1+\ell'_2}$. Thus, the claim follows by Lemma 3.5. ∎

Lemma 4.3. *Let* $\mathfrak{G}^{\mathrm{col}}_{k\ell} \equiv^M_r \mathfrak{G}^{\mathrm{col}}_{k\ell'}$. *Then* $\mathfrak{G}_{k\ell} \equiv^M_r \mathfrak{G}_{k\ell'}$.

Proof. For each fixed horizontal dimension k, there exists a one-dimensional quantifier-free interpretation of a cliquey grid in its respective horizontally coloured counterpart since we can define the horizontal equivalence relation $\sim_h$ in terms of the colours as follows:

$$x \sim_h y \;\equiv\; \bigvee_{i=1}^{k} P_i x \wedge P_i y\,.$$

■

Actually, the argument implies that Duplicator wins a game on cliquey grids using the same strategy that is winning in the corresponding game on coloured grids since a strategy preserving the colours of selected elements especially preserves the equivalence relation $\sim_h$.

Before stating the main lemma, we will first prove a combinatorial result which will later help Duplicator in synthesising her winning strategy and introduce the following weakened notion of equality between numbers.

Definition 4.4. Two numbers $a, b \in \mathbb{N}$ are called *threshold t equal (mod M)*, denoted $a =_t^M b$, if

(i) $a = b$ or

(ii) $a, b \geq t$ and $a \equiv b \pmod{M}$.

Intuitively, $a =_t^M b$ means that the numbers are equal if they are small, or that they are at least congruent modulo all $m \in M$ if they are both at least as large as the threshold t.

Lemma 4.5. *For every $p, t \in \mathbb{N}$, and $M \subseteq_{fin} \mathbb{N}^+$, we can choose an arbitrary $T \geq p \cdot (t + \operatorname{lcm}(M) - 1)$ such that for all sets A and B with $|A| =_T^M |B|$ and for every equivalence relation $\approx_A$ on A of index at most p there exists an equivalence relation $\approx_B$ on B and a bijection $g\colon A/_{\approx_A} \to B/_{\approx_B}$ satisfying $|\{a' \in A : a \approx_A a'\}| =_t^M |g(\{a' \in A : a \approx_A a'\})|$ for all $a \in A$.*

Proof. We let $\{a_1, \ldots, a_{p'}\}$, where $p' \leq p$ denotes the index of $\approx_A$, be the set of class representatives of $A/_{\approx_A}$, and we let $[a]_{\approx_A} := \{a' \in A : a' \approx_A a\}$ denote the equivalence class of a in A. Note that we will usually omit the subscript $\approx_A$ if it is clear from the context and instead reserve the letters a and b for elements denoting equivalence classes in A and B, respectively. Furthermore, a set will be called *small* in the following if it contains less than t elements and *large* otherwise.

The equivalence relation $\approx_B$ on B is constructed by partitioning the set into p' disjoint non-empty subsets $\{B_1, \ldots, B_{p'}\}$ as follows. If $|A| = |B|$, for each class $[a_i]$, we choose a set B_i with exactly $|[a_i]|$ many elements. If $|A|, |B| \geq T$, we have to distinguish between the treatment of small and large classes. Since $|A| \geq T \geq p \cdot (t + \operatorname{lcm}(M) - 1)$, $\operatorname{lcm}(M) \geq 1$, and the index of $\approx_A$ is at most p, at least one of the equivalence classes contains at least t elements, i.e., it is large, and without loss of generality, it is assumed that this is the case for $[a_1]$. For each small class $[a_i]$, we choose a set B_i with exactly $|[a_i]|$ many elements. If $[a_i]$ is large, we choose a set B_i containing $t + \ell$ many elements where ℓ is the smallest non-negative integer such that $|[a_i]| \equiv |B_i| \pmod{M}$. The number of elements selected according to these rules is at most $p \cdot (t + \operatorname{lcm}(M) - 1) \leq T \leq |B|$. Since $[a_1]$ is large by assumption, any possibly remaining elements in B, that have not been assigned to one of the subsets $B_1, \ldots, B_{p'}$ yet, can be safely added to B_1 without violating the condition that $|[a_1]| \equiv |B_1| \pmod{M}$.

This partitioning uniquely defines the equivalence relation $\approx_B := \bigcup_{i=1}^{p'} (B_i \times B_i)$ on B. By selecting an arbitrary element of each B_i we get a set of class representatives $\{b_1, \ldots, b_{p'}\}$ which directly yields the bijection $g\colon [a_i] \mapsto [b_i]$ for all $1 \leq i \leq p'$ satisfying $|[a]| =_t^M |g([a])|$ for all $a \in A$ by construction. ■

The following lemma extends the results on CMSO-equivalence of *large enough sets* to *large enough grids* by giving a sufficient condition on the sizes of two grids for the existence of a winning strategy for Duplicator in an r-round (mod M) game on the two structures.

Due to the inductive nature of the proof that involves, in each step, a construction of equivalence classes as in the above lemma, we need as a criterion for the size, for fixed $p \in \mathbb{N}$ and $M \subseteq_{\text{fin}} \mathbb{N}^+$, a function $f_{p,M} : \mathbb{N} \to \mathbb{N}$ such that, for all $r \in \mathbb{N}^+$ and $t = f_{p,M}(r-1)$, we can choose $T = f_{p,M}(r)$ in the previous lemma. One function satisfying, for all $r \in \mathbb{N}^+$, the inequality $f_{p,M}(r) \geq p \cdot (f_{p,M}(r-1) + \operatorname{lcm}(M) - 1)$ derived from the condition imposed on T is $f_{p,M}(r) = 2 \cdot (p^r - 1) \cdot \operatorname{lcm}(M)$.

Lemma 4.6. *Let $M \subseteq_{fin} \mathbb{N}^+$, $r \in \mathbb{N}$ and $k > 1$ be fixed. Then for $f(r) := f_{2^k,M}(r) = (2^{kr+1} - 2)\operatorname{lcm}(M)$, as given above, $\mathfrak{G}_{k\ell_1} \equiv_r^M \mathfrak{G}_{k\ell_2}$ if $\ell_1 =_{f(r)}^M \ell_2$.*

Proof. As motivated by Lemma 4.3, we consider the r-round (mod M) Ehrenfeucht-Fraïssé game on the corresponding horizontally coloured cliquey grids $\mathfrak{G}_{k\ell_1}^{\text{col}}$ and $\mathfrak{G}_{k\ell_2}^{\text{col}}$, and we show by induction on the number of rounds that Duplicator has a winning strategy in this game.

Intuitively, the proof proceeds as follows. Spoiler's set move induces an equivalence relation on the set of columns forming the grid he plays in, and the previous lemma implies that Duplicator is able to construct an equivalence relation on the columns of the other grid which is similar in the sense that corresponding equivalence classes satisfy certain cardinality constraints. Since the grids can be regarded as disjoint unions of these equivalence classes, we can argue by induction that corresponding subparts of the two grids, being similar enough, cannot be distinguished during the remaining $r-1$ rounds of the game.

The case where $\ell_1 = \ell_2$ is trivial since grids of the same dimensions are isomorphic. Thus, we assume in the following that $\ell_1, \ell_2 \geq f(r)$ and $\ell_1 \equiv \ell_2 \pmod{M}$. The claim is obviously true for $r = 0$, hence we assume that it holds for $r-1$ and proceed with the inductive step. As before, we assume without loss of generality that Spoiler makes his moves in $\mathfrak{G}_{k\ell_1}$ since the other case is symmetric.

A *coloured k-column* is a $\{\sim_v, P_1, \dots, P_k\}$-structure isomorphic to $\mathfrak{C}_k^{\text{col}} := \mathfrak{G}_{k,1}^{\text{col}}$, such that a coloured grid can be regarded as a disjoint union of columns. Given a subset S of vertices of a grid and one of its coloured k-columns $\mathfrak{C}$ with universe C, the *colour-type of $\mathfrak{C}$ induced by S* is defined as the isomorphism type of the expansion $(\mathfrak{C}, S \cap C)$ denoted by $\operatorname{tp}(\mathfrak{C}, S)$. Given a set $\mathcal{F}$ of k-columns, each subset S of all of their vertices gives rise to an equivalence relation $\approx_S$ on $\mathcal{F}$ by virtue of $\mathfrak{C}_1 \approx_S \mathfrak{C}_2$ if, and only if, $\operatorname{tp}(\mathfrak{C}_1, S) = \operatorname{tp}(\mathfrak{C}_2, S)$. Note that the index of $\approx_S$ is at most 2^k.

Assume, Spoiler performs a set move and chooses a subset S in $\mathfrak{G}_{k\ell_1}^{\text{col}} = \mathfrak{C}_1 \dot\cup \cdots \dot\cup \mathfrak{C}_{\ell_1}$. As described above, S induces an equivalence relation $\approx_S$ with at most 2^k equivalence classes on the set $\mathcal{F} = \{\mathfrak{C}_1, \dots, \mathfrak{C}_{\ell_1}\}$ of columns forming the grid. For $p = 2^k$, $t = f(r-1)$ and M as given, by the previous lemma, there is an equivalence relation $\approx'_S$ on the set $\mathcal{F}' = \{\mathfrak{C}'_1, \dots, \mathfrak{C}'_{\ell_2}\}$ of columns on the Duplicator's grid $\mathfrak{G}_{k\ell_2}^{\text{col}}$ since $\ell_1, \ell_2 \geq f(r)$. Furthermore, there is a bijection g mapping equivalence classes of columns in one grid to the other.

Given that the index of both $\approx_S$ and $\approx'_S$ is $p' \leq p = 2^k$, we can assume $\{\mathfrak{C}_1, \dots, \mathfrak{C}_{p'}\}$ and $\{\mathfrak{C}'_1, \dots, \mathfrak{C}'_{p'}\}$ to be the sets of class representatives of $\approx_S$ and $\approx'_S$, respectively. Duplicator now selects the unique set D of elements such that $\operatorname{tp}(\mathfrak{C}, S) = \operatorname{tp}(\mathfrak{C}', D)$ for all $1 \leq i \leq p'$, $\mathfrak{C} \in [\mathfrak{C}_i]$ and $\mathfrak{C}' \in g([\mathfrak{C}_i])$.

For each $1 \leq i \leq p'$, we let $\langle \mathfrak{C}_i \rangle := \mathfrak{G}_{k\ell_1}^{\text{col}} \restriction [\mathfrak{C}_i]$ and $\langle \mathfrak{C}'_i \rangle := \mathfrak{G}_{k\ell_2}^{\text{col}} \restriction [\mathfrak{C}'_i]$ denote the substructures of the grids $\mathfrak{G}_{k\ell_1}^{\text{col}}$ and $\mathfrak{G}_{k\ell_2}^{\text{col}}$ induced by the sets of columns $[\mathfrak{C}_i]$ and $[\mathfrak{C}'_i]$, respectively. By construction, we have $|[\mathfrak{C}_i]| =_{f(r-1)}^M |[\mathfrak{C}'_i]|$ for all i. Thus, depending on whether $[\mathfrak{C}_i]$ (and hence $[\mathfrak{C}'_i]$) are small or large with respect to the threshold $f(r-1)$, either $\langle \mathfrak{C}_i \rangle \cong \langle \mathfrak{C}'_i \rangle$ or $\langle \mathfrak{C}_i \rangle \equiv_{r-1}^M \langle \mathfrak{C}'_i \rangle$ by the induction hypothesis. Since S and D induce the

same colour-types on the columns in $[\mathfrak{C}_i]$ and $[\mathfrak{C}'_i]$, respectively, we have

$$\big(\langle \mathfrak{C}_i \rangle, S \cap \mathrm{univ}(\langle \mathfrak{C}_i \rangle)\big) \equiv^M_{r-1} \big(\langle \mathfrak{C}'_i \rangle, D \cap \mathrm{univ}(\langle \mathfrak{C}'_i \rangle)\big)$$

for all i, where $\mathrm{univ}(\cdot)$ denotes the universe of the respective structure. Thus, iterating Lemma 3.5 yields that Duplicator has a winning strategy in the remaining rounds of the game $\mathcal{G}^M_{r-1}(\mathfrak{G}^{\mathrm{col}}_{k\ell_1}, \mathfrak{G}^{\mathrm{col}}_{k\ell_2})$ from position (S, D).

If Spoiler makes a point move s, say in column $\mathfrak{C}_1$ of the grid $\mathfrak{G}^{\mathrm{col}}_{k\ell_1}$, Duplicator picks an arbitrary element d of the same colour in her grid, say in column $\mathfrak{C}'_1$. As the substructures consisting of just the columns containing the chosen elements are isomorphic, i.e., $\big(\mathfrak{C}_1, s\big) \cong \big(\mathfrak{C}'_1, d\big)$, and by the induction hypothesis we have $\mathfrak{C}_2 \dot\cup \cdots \dot\cup \mathfrak{C}_{\ell_1} \equiv^M_{r-1} \mathfrak{C}'_2 \dot\cup \cdots \dot\cup \mathfrak{C}'_{\ell_2}$, Duplicator can win the remaining $(r-1)$-round game from position (s, d) by Lemma 3.5. ∎

Now we have the necessary tools available to prove the main theorem.

Theorem 4.7. $\mathrm{CMSO} \subsetneq \mathrm{MSO}[<]_{inv}$.

Proof. We show that the class $\mathcal{C} := \{\, \mathfrak{G}_{k\ell} \,:\, k|\ell \,\}$ is not definable in CMSO but order-invariantly definable in MSO by the sentence $\psi_{\mathrm{grid}} \wedge \varphi$, where

$$\varphi = \exists \min \exists c \left(\begin{array}{c} \forall x(\min \leq x) \wedge \neg \exists z(E_h(c,z) \vee E_v(c,z)) \\ \wedge\ \forall T\big(\forall x \forall y(Tx \wedge \varphi_{\mathrm{diag}}(x,y) \to Ty) \wedge T\min \ \to Tc\big) \end{array} \right),$$

and

$$\begin{aligned} \varphi_{\mathrm{diag}}(x,y) = {} & \big(\exists z(E_v(x,z) \wedge E_h(z,y))\big) \\ & \vee \big(\neg \exists z E_v(x,z) \wedge \exists z(z \sim_h \min \wedge\, z \sim_v x \wedge E_h(z,y))\big)\,, \\ E_h(x,y) = {} & x \sim_h y \wedge \exists x_0 \exists y_0 \left(\begin{array}{l} x_0 \sim_h \min \wedge\, y_0 \sim_h \min \\ \wedge\ x \sim_v x_0 \wedge y \sim_v y_0 \wedge x_0 < y_0 \\ \wedge\ \forall z_0(z_0 \sim_h \min \ \to z_0 \leq x_0 \vee z_0 \geq y_0) \end{array} \right), \\ E_v(x,y) = {} & x \sim_v y \wedge \exists x_0 \exists y_0 \left(\begin{array}{l} x_0 \sim_v \min \wedge\, y_0 \sim_v \min \\ \wedge\ x \sim_h x_0 \wedge y \sim_h y_0 \wedge x_0 < y_0 \\ \wedge\ \forall z_0(z_0 \sim_v \min \ \to z_0 \leq x_0 \vee z_0 \geq y_0) \end{array} \right). \end{aligned}$$

As hinted above, the horizontal and vertical edge relations (E_h and E_v, respectively) are defined using the successor relation which is induced by an arbitrary ordering on the row (and column) containing the minimal element (min) which itself serves as the lower left corner of the grid. φ_{diag} defines diagonal steps through the grid that wrap around from the top to the bottom row. Finally, φ states that the pair consisting of the lower left corner (min) and the upper right corner (c) of the grid is contained in the transitive closure of φ_{diag}. Obviously, there is such a sawtooth-shaped path starting at min and ending exactly in the upper right corner if, and only if, $k|\ell$.

The second step consists in showing that $\mathcal{C}$ is not definable in CMSO. Towards this goal, we show that for any choice of $r \in \mathbb{N}$ and $M \subseteq_{\mathrm{fin}} \mathbb{N}^+$, we can find $k, \ell_1, \ell_2 \in \mathbb{N}$, such that $\mathfrak{G}_{k\ell_1} \in \mathcal{C}$, $\mathfrak{G}_{k\ell_2} \notin \mathcal{C}$, and $\mathfrak{G}_{k\ell_1} \equiv^M_r \mathfrak{G}_{k\ell_2}$ which contradicts the CMSO-definability of $\mathcal{C}$.

Let $r \in \mathbb{N}$ and $M \subseteq_{\mathrm{fin}} \mathbb{N}^+$ be fixed. We choose $s \geq r+1$ such that $2^s \nmid \mathrm{lcm}(M)$. Let $k = 2^s$, $\ell_1 = 2^{kr+1}\,\mathrm{lcm}(M)$, and $\ell_2 = \ell_1 + \mathrm{lcm}(M)$. Obviously, ℓ_1 and ℓ_2 satisfy the conditions of Lemma 4.6, and thus $\mathfrak{G}_{k\ell_1} \equiv^M_r \mathfrak{G}_{k\ell_2}$.

Furthermore, $\ell_1 = k \cdot 2^{2^s \cdot r - s + 1} \operatorname{lcm}(M)$, hence $k \mid \ell_1$ and $\mathfrak{G}_{k\ell_1} \in \mathcal{C}$. On the other hand, $k \nmid \ell_2 = \ell_1 + \operatorname{lcm}(M)$ by the choice of s, thus $\mathfrak{G}_{k\ell_2} \notin \mathcal{C}$. ∎

5. Conclusion

We have provided a characterisation of the expressiveness of CMSO in terms of an Ehrenfeucht-Fraïssé game that naturally extends the known game capturing MSO-definability, and we have presented a class of structures that are shown, using the proposed game characterisation, to be undefinable by a CMSO-sentence yet being definable by an order-invariant MSO-sentence. This establishes that order-invariant MSO is strictly more expressive than counting MSO in the finite. Modifying the separating example by considering a variant of cliquey grids where the two separate equivalence relations are unified into a single binary relation and considering, e.g., the class of such grids where the horizontal dimension exactly matches the vertical dimension, we can also confirm Courcelle's original conjecture.

Corollary 5.1. CMSO-*definability is strictly weaker than* $\mathrm{MSO}[<]_{inv}$-*definability for general graphs.*

The separating query being essentially a transitive closure query, i.e., the only place where monadic second-order quantification is used is in the definition of the transitive closure of a binary relation, we can conclude that the same class of structures yields a separation of $(\mathrm{D})\mathrm{TC}^1[<]_{inv}$ from $(\mathrm{D})\mathrm{TC}^1$ (the extension of FO by a (deterministic) transitive closure operator on binary relations) and even from $(\mathrm{D})\mathrm{TC}^1$ extended with modulo-counting predicates since $(\mathrm{D})\mathrm{TC}^1 \subseteq \mathrm{MSO}$. Finding separating examples concerning higher arity (D)TC or even full (D)TC requires further investigation since, in general, $\mathrm{MSO} \subsetneq \mathrm{DTC}^2$.

Following an opposite line of research, it would be interesting to identify further classes of graphs, besides classes of graphs of bounded tree-width, on which $\mathrm{MSO}[<]_{inv}$ is no more expressive than CMSO.

References

[BS05] Michael Benedikt and Luc Segoufin. Towards a characterization of order-invariant queries over tame structures. In *Proc. 14th Conf. on Computer Science Logic, CSL 2005*, pp. 276–291, 2005.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.

[Cou96] Bruno Courcelle. The monadic second-order logic of graphs X: Linear orderings. *Theoretical Computer Science*, 160:87–143, 1996.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1995.

[Lap98] Denis Lapoire. Recognizability equals monadic second-order definability for sets of graphs of bounded tree-width. In *Proc. STACS 1998*, pp. 618–628, 1998.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[Nur00] Juha Nurmonen. Counting modulo quantifiers on finite structures. *Information and Computation*, 160(1-2):62–87, 2000.

[Ott95] Martin Otto. A note on the number of monadic quantifiers in monadic Σ^1_1. *Information Processing Letters*, 53(6):337–339, March 1995.

[Ott00] Martin Otto. Epsilon-logic is more expressive than first-order logic over finite structures. *Journal of Symbolic Logic*, 65(4):1749–1757, 2000.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 325-336
www.stacs-conf.org

SUCCINCTNESS OF THE COMPLEMENT AND INTERSECTION OF REGULAR EXPRESSIONS

WOUTER GELADE AND FRANK NEVEN

Hasselt University and Transnational University of Limburg, School for Information Technology
E-mail address: firstname.lastname@uhasselt.be

Abstract. We study the succinctness of the complement and intersection of regular expressions. In particular, we show that when constructing a regular expression defining the complement of a given regular expression, a double exponential size increase cannot be avoided. Similarly, when constructing a regular expression defining the intersection of a fixed and an arbitrary number of regular expressions, an exponential and double exponential size increase, respectively, can in worst-case not be avoided. All mentioned lower bounds improve the existing ones by one exponential and are tight in the sense that the target expression can be constructed in the corresponding time class, i.e., exponential or double exponential time. As a by-product, we generalize a theorem by Ehrenfeucht and Zeiger stating that there is a class of DFAs which are exponentially more succinct than regular expressions, to a fixed four-letter alphabet. When the given regular expressions are one-unambiguous, as for instance required by the XML Schema specification, the complement can be computed in polynomial time whereas the bounds concerning intersection continue to hold. For the subclass of single-occurrence regular expressions, we prove a tight exponential lower bound for intersection.

1. Introduction

The two central questions addressed in this paper are the following. Given regular expressions $r, r_1, \ldots, r_k$ over an alphabet Σ,

(1) what is the complexity of constructing a regular expression $r_\neg$ defining $\Sigma^* \setminus L(r)$, that is, the complement of r?
(2) what is the complexity of constructing a regular expression $r_\cap$ defining $L(r_1) \cap \cdots \cap L(r_k)$?

In both cases, the naive algorithm takes time double exponential in the size of the input. Indeed, for the complement, transform r to an NFA and determinize it (first exponential step), complement it and translate back to a regular expression (second exponential step). For the intersection there is a similar algorithm through a translation to NFAs, taking the crossproduct and a retranslation to a regular expression. Note that both algorithms do not only take double exponential time but also result in a regular expression of double exponential size. In this paper, we exhibit classes of regular expressions for which this double

Wouter Gelade is a Research Assistant of the Fund for Scientific Research - Flanders (Belgium).

exponential size increase cannot be avoided. Furthermore, when the number k of regular expressions is fixed, $r_\cap$ can be constructed in exponential time and we prove a matching lower bound for the size increase. In addition, we consider the fragments of one-unambiguous and single-occurrence regular expressions relevant to XML schema languages [2, 3, 13, 23]. Our main results are summarized in Table 1.

The main technical part of the paper is centered around the generalization of a result by Ehrenfeucht and Zeiger [8]. They exhibit a class of languages $(Z_n)_{n\in\mathbb{N}}$ each of which can be accepted by a DFA of size $\mathcal{O}(n^2)$ but cannot be defined by a regular expression of size smaller than 2^{n-1}. The most direct way to define Z_n is by the DFA that accepts it: the DFA is a graph consisting of n states, labeled 0 to $n-1$, which are fully connected and the edge between state i and j carries the label $a_{i,j}$. It now accepts all paths in the graph, that is, all strings of the form $a_{i_0,i_1}a_{i_1,i_2}\cdots a_{i_k,i_{k+1}}$. Note that the alphabet over which Z_n is defined grows quadratically with n. We generalize their result to a four-letter alphabet. In particular, we define K_n as the binary encoding of Z_n using a suitable encoding for $a_{i,j}$ and prove that every regular expression defining K_n should be at least of size 2^n. As integers are encoded in binary the complement and intersection of regular expressions can now be used to separately encode K_{2^n} (and slight variations thereof) leading to the desired results. In [9] the same generalization as obtained here is attributed to Waizenegger [35]. Unfortunately, we believe that proof to be incorrect as we discuss in the full version of this paper.

Although the succinctness of various automata models have been investigated in depth [14] and more recently those of logics over (unary alphabet) strings [15], the succinctness of regular expressions has hardly been addressed. For the complement of a regular expression an exponential lower bound is given by Ellul et al [9]. For the intersection of an arbitrary number of regular expressions Petersen gave an exponential lower bound [28], while Ellul et al [9] mention a quadratic lower bound for the intersection of two regular expressions. In fact, in [9], it is explicitly asked what the maximum achievable blow-up is for the complement of one and the intersection of two regular expressions (Open Problems 4 and 5). Although we do not answer these questions in the most precise way, our lower bounds improve the existing ones by one exponential and are tight in the sense that the target expression can be constructed in the time class matching the space complexity of the lower bounds.

Succinctness of complement and intersection relate to the succinctness of semi-extended (RE($\cap$)) and extended regular expressions (RE($\cap$,$\neg$)). These are regular expressions augmented with intersection and both complement and intersection operators, respectively. Their membership problem has been extensively studied [18, 20, 26, 28, 30]. Furthermore, non-emptiness and equivalence of RE($\cap$,$\neg$) is non-elementary [33]. For RE($\cap$), inequivalence is EXPSPACE-complete [10, 16, 29], and non-emptiness is PSPACE-complete [10, 16] even when restricted to the intersection of a (non-constant) number of regular expressions [19]. Several of these papers hint upon the succinctness of the intersection operator and provide dedicated techniques in dealing with the new operator directly rather than through a translation to ordinary regular expressions [20, 28]. Our results present a double exponential lower bound in translating RE($\cap$) to RE and therefore justify even more the development for specialized techniques.

A final motivation for this research stems from its application in the emerging area of XML-theory [21, 27, 31, 34]. From a formal language viewpoint, XML documents can be seen as labeled unranked trees and collections of these documents are defined by schemas. A schema can take various forms, but the most common ones are Document Type Definitions (DTDs) [4] and XML Schema Definitions (XSDs) [32] which are grammar based formalisms

	complement	intersection (fixed)	intersection (arbitrary)
regular expression	2-exp	exp	2-exp
one-unambiguous	poly	exp	2-exp
single-occurrence	poly	exp	exp

Table 1: Overview of the size increase for the various operators and subclasses. All non-polynomial complexities are tight.

with regular expressions at right-hand sides of rules [23, 25]. Many questions concerning schemas reduce to corresponding questions on the classes of regular expressions used as right-hand sides of rules as is exemplified for the basic decision problems studied in [11] and [22]. Furthermore, the lower bounds presented here are utilized in [12] to prove, among other things, lower bounds on the succinctness of existential and universal pattern-based schemas on the one hand, and single-type EDTDs (a formalization of XSDs) and DTDs, on the other hand. As the DTD and XML Schema specification require regular expressions occurring in rules to be *deterministic*, formalized by Brüggemann-Klein and Wood in terms of one-unambiguous regular expressions [6], we also investigate the complement and intersection of those. In particular, we show that a one-unambiguous regular expressions can be complemented in polynomial time, whereas the lower bounds concerning intersection carry over from unrestricted regular expressions. A study in [2] reveals that most of the one-unambiguous regular expression used in practice take a very simple form: every alphabet symbol occurs at most once. We refer to those as single-occurrence regular expressions (SOREs) and show a tight exponential lower bound for intersection.

Outline. In Section 2, we introduce the necessary notions concerning (one-unambiguous) regular expressions and automata. In Section 3, we extend the result by Ehrenfeucht and Zeiger to a fixed alphabet using the family of languages $(K_n)_{n\in\mathbb{N}}$. In Section 4, we consider the succinctness of complement. In Section 5, we consider the succinctness of intersection of several classes of regular expressions. We conclude in Section 6. A version of this paper containing all proofs is available from the authors' webpages.

2. Preliminaries

2.1. Regular expressions

By $\mathbb{N}$ we denote the natural numbers without zero. For the rest of the paper, Σ always denotes a finite alphabet. A Σ-*string* (or simply string) is a finite sequence $w = a_1 \cdots a_n$ of Σ-symbols. We define the length of w, denoted by $|w|$, to be n. We denote the empty string by ε. The set of *positions of* w is $\{1, \ldots, n\}$ and the *symbol of* w *at position* i is a_i. By $w_1 \cdot w_2$ we denote the *concatenation* of two strings w_1 and w_2. As usual, for readability, we denote the concatenation of w_1 and w_2 by $w_1 w_2$. The set of all strings is denoted by Σ^* and the set of all non-empty strings by Σ^+. A *string language* is a subset of Σ^*. For two string languages $L, L' \subseteq \Sigma^*$, we define their concatenation $L \cdot L'$ to be the set $\{w \cdot w' \mid w \in L, w' \in L'\}$. We abbreviate $L \cdot L \cdots L$ (i times) by L^i.

The set of *regular expressions* over Σ, denoted by RE, is defined in the usual way: $\emptyset$, ε, and every Σ-symbol is a regular expression; and when r_1 and r_2 are regular expressions, then $r_1 \cdot r_2$, $r_1 + r_2$, and r_1^* are also regular expressions.

By RE($\cap$,$\neg$) we denote the class of *extended regular expressions*, that is, RE extended with intersection and complementation operators. So, when r_1 and r_2 are RE($\cap$,$\neg$)-expressions then so are $r_1 \cap r_2$ and $\neg r_1$. By RE($\cap$) and RE($\neg$) we denote RE extended solely with the intersection and complement operator, respectively.

The language defined by an extended regular expression r, denoted by $L(r)$, is inductively defined as follows: $L(\emptyset) = \emptyset$; $L(\varepsilon) = \{\varepsilon\}$; $L(a) = \{a\}$; $L(r_1 r_2) = L(r_1) \cdot L(r_2)$; $L(r_1 + r_2) = L(r_1) \cup L(r_2)$; $L(r^*) = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} L(r)^i$; $L(r_1 \cap r_2) = L(r_1) \cap L(r_2)$; and $L(\neg r_1) = \Sigma^* \setminus L(r_1)$.

By $\bigcup_{i=1}^{k} r_i$, and r^k, with $k \in \mathbb{N}$, we abbreviate the expression $r_1 + \cdots + r_k$, and $rr \cdots r$ (k-times), respectively. For a set $S = \{a_1, \ldots, a_n\} \subseteq \Sigma$, we abbreviate by S the regular expression $a_1 + \cdots + a_n$.

We define the *size* of an extended regular expression r over Σ, denoted by $|r|$, as the number of Σ-symbols and operators occurring in r disregarding parentheses. This is equivalent to the length of its (parenthesis-free) reverse Polish form [37]. Formally, $|\emptyset| = |\varepsilon| = |a| = 1$, for $a \in \Sigma$, $|r_1 r_2| = |r_1 \cap r_2| = |r_1 + r_2| = |r_1| + |r_2| + 1$, and $|\neg r| = |r^*| = |r| + 1$.

Other possibilities considered in the literature for defining the size of a regular expression are: (1) counting all symbols, operators, and parentheses [1, 17]; or, (2) counting only the Σ-symbols. However, Ellul et al. [9] have shown that for regular expressions (so, without $\neg$ and $\cap$), provided they are preprocessed by syntactically eliminating superfluous $\emptyset$- and ε-symbols, and nested stars, the three length measures are identical up to a constant multiplicative factor. For extended regular expressions, counting only the Σ-symbols is not sufficient, since for instance the expression $(\neg\varepsilon)(\neg\varepsilon)(\neg\varepsilon)$ does not contain any Σ-symbols. Therefore, we define the size of an expression as the length of its reverse Polish form.

2.2. One-unambiguous regular expressions and SOREs

As mentioned in the introduction, several XML schema languages restrict regular expressions occurring in rules to be *deterministic*, formalized by Brüggemann-Klein and Wood [6] in terms of one-unambiguity. We introduce this notion next.

To indicate different occurrences of the same symbol in a regular expression, we mark symbols with subscripts. For instance, the *marking* of $(a + b)^* a + bc$ is $(a_1 + b_2)^* a_3 + b_4 c_5$. We denote by $r^\flat$ the marking of r and by $\mathrm{Sym}(r^\flat)$ the subscripted symbols occurring in $r^\flat$. When r is a marked expression, then $r^\natural$ over Σ is obtained from r by dropping all subscripts. This notion is extended to words and languages in the usual way.

Definition 2.1. A regular expression r is *one-unambiguous* iff for all strings $w, u, v \in \mathrm{Sym}(r^\flat)^*$, and all symbols $x, y \in \mathrm{Sym}(r^\flat)$, the conditions $uxv, uyw \in L(r^\flat)$ and $x \neq y$ imply $x^\natural \neq y^\natural$.

For instance, the regular expression $r = a^* a$, with marking $r^\flat = a_1^* a_2$, is not one-unambiguous. Indeed, the marked strings $a_1 a_2$ and $a_1 a_1 a_2$ both in $L(r^\flat)$ do not satisfy the conditions in the previous definition. The equivalent expression aa^*, however, is one-unambiguous. The intuition behind the definition is that positions in the input string can be matched in a deterministic way against a one-unambiguous regular expression without looking ahead. For instance, for the expression aa^*, the first a of an input string is always matched against the leading a in the expression, while every subsequent a is matched against the last a. Unfortunately, one-unambiguous regular languages do not form a very robust class as they are not even closed under the Boolean operations [6].

The following subclass captures the class of regular expressions occurring in XML schemas on the Web [2]:

Definition 2.2. A *single-occurrence regular expression (SORE)* is a regular expression where every alphabet symbol occurs at most once. In addition, we allow the operator r^+ which defines rr^*.

For instance, $(a + b)^+c$ is a SORE while $a^*(a + b)^+$ is not. Clearly, every SORE is one-unambiguous. Note that SOREs define local languages and that over a fixed alphabet there are only finitely many of them.

2.3. Finite automata

A non-deterministic finite automaton (NFA) A is a 4-tuple (Q, q_0, δ, F) where Q is the set of states, q_0 is the initial state, F is the set of final states and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. We write $q \Rightarrow_{A,w} q'$ when w takes A from state q to q'. So, w is accepted by A if $q_0 \Rightarrow_{A,w} q'$ for some $q' \in F$. The set of strings accepted by A is denoted by $L(A)$. The size of an NFA is $|Q| + |\delta|$. An NFA is *deterministic* (or a DFA) if for all $a \in \Sigma, q \in Q$, $|\{(q, a, q') \in \delta \mid q' \in Q\}| \leq 1$.

We make use of the following known results.

Theorem 2.3. *Let $A_1, \ldots, A_m$ be NFAs over Σ with $|A_i| = n_i$ for $i \leq m$, and $|\Sigma| = k$.*

(1) A regular expression r, with $L(r) = L(A_1)$, can be constructed in time $\mathcal{O}(m_1 k 4^{m_1})$, where m_1 is the number of states of A_1 [24, 9].

(2) A DFA B with 2^{n_1} states, such that $L(B) = L(A_1)$, can be constructed in time $\mathcal{O}(2^{n_1})$ [36].

(3) A DFA B with 2^{n_1} states, such that $L(B) = \Sigma^* \setminus L(A_1)$, can be constructed in time $\mathcal{O}(2^{n_1})$ [36].

(4) Let $r \in RE$. An NFA B with $|r|+1$ states, such that $L(B) = L(r)$, can be constructed in time $\mathcal{O}(|r| \cdot |\Sigma|)$ [5].

(5) Let $r \in RE(\cap)$. An NFA B with $2^{|r|}$ states, such that $L(B) = L(r)$, can be constructed in time exponential in the size of r [10].

3. A generalization of a Theorem by Ehrenfeucht and Zeiger to a fixed alphabet

We first introduce the family $(Z_n)_{n\in\mathbb{N}}$ defined by Ehrenfeucht and Zeiger over an alphabet whose size grows quadratically with the parameter n [8]:

Definition 3.1. Let $n \in \mathbb{N}$ and $\Sigma_n = \{a_{i,j} \mid 0 \leq i, j \leq n - 1\}$. Then, Z_n contains exactly all strings of the form $a_{i_0,i_1} a_{i_1,i_2} \cdots a_{i_{k-1},i_k}$ where $k \in \mathbb{N}$.

A way to interpret Z_n is to consider the DFA with states $\{0, \ldots, n - 1\}$ which is fully connected and where the edge between state i and j is labeled with $a_{i,j}$. The language Z_n then consists of all paths in the DFA. [1]

Ehrenfeucht and Zeiger obtained the succinctness of DFAs with respect to regular expressions through the following theorem:

[1] Actually, in [8], only paths from state 0 to state $n - 1$ are considered. We use our slightly modified definition as it will be easier to generalize to a fixed arity alphabet suited for our purpose in the sequel.

Theorem 3.2 ([8])**.** *For $n \in \mathbb{N}$, any regular expression defining Z_n must be of size at least 2^{n-1}. Furthermore, there is a DFA of size $\mathcal{O}(n^2)$ accepting Z_n.*

Our language K_n is then the straightforward binary encoding of Z_n that additionally swaps the pair of indices in every symbol $a_{i,j}$. Thereto, for $a_{i,j} \in \Sigma_n$, define the function ρ_n as

$$\rho_n(a_{i,j}) = \text{enc}(j)\$\text{enc}(i)\#,$$

where enc(i) and enc(j) denote the $\lceil \log(n) \rceil$-bit binary encodings of i and j, respectively. Note that since $i, j < n$, i and j can be encoded using only $\lceil \log(n) \rceil$-bits. We extend the definition of ρ_n to strings in the usual way: $\rho_n(a_{i_0,i_1} \cdots a_{i_{k-1},i_k}) = \rho_n(a_{i_0,i_1}) \cdots \rho_n(a_{i_{k-1},i_k})$.

We are now ready to define K_n.

Definition 3.3. Let $\Sigma_K = \{0, 1, \$, \#\}$. For $n \in \mathbb{N}$, let $K_n = \{\rho_n(w) \mid w \in Z_n\}$.

For instance, for $n = 5$, $w = a_{3,2}a_{2,1}a_{1,4}a_{4,2} \in Z_5$ and thus

$$\rho_n(w) = 010\$011\#001\$010\#100\$001\#010\$100\# \in K_5.$$

We generalize the previous theorem as follows:

Theorem 3.4. *For any $n \in \mathbb{N}$, with $n \geq 2$,*

(1) any regular expression defining K_n is of size at least 2^n; and,
(2) there is a DFA A_n of size $\mathcal{O}(n^2 \log n)$ defining K_n.

The construction of A_n is omitted. The rest of this section is devoted to the proof of Theorem 3.4(1). It follows the structure of the proof of Ehrenfeucht and Zeiger but is technically more involved as it deals with binary encodings of integers.

We start by introducing some terminology. Let $w = a_{i_0,i_1}a_{i_1,i_2} \cdots a_{i_{k-1},i_k} \in Z_n$. We say that i_0 is the *start-point* of w and i_k is its *end-point*. Furthermore, we say that w *contains* i or i *occurs in* w if i occurs as an index of some symbol in w. That is, $a_{i,j}$ or $a_{j,i}$ occurs in w for some j. For instance, $a_{0,2}a_{2,2}a_{2,1} \in Z_5$, has start-point 0, end-point 1, and contains 0, 1 and 2. The notions of contains, occurs, start- and end-point of a string w are also extended to K_n. So, the start and end-points of $\rho_n(w)$ are the start and end-points of w, and w contains the same integers as $\rho_n(w)$.

For a regular expression r, we say that i is a *sidekick* of r when it occurs in every non-empty string defined by r. A regular expression s is a starred subexpression of a regular expression r when s is a subexpression of r and is of the form t^*.

Now, the following lemma holds:

Lemma 3.5. *Any starred subexpression s of a regular expression r defining K_n has a sidekick.*

We now say that a regular expression r is *normal* if every starred subexpression of r has a sidekick. In particular, any expression defining K_n is normal. We say that a regular expression r *covers* a string w if there exist strings $u, u' \in \Sigma^*$ such that $uwu' \in L(r)$. If there is a greatest integer m for which r covers w^m, we call m the *index* of w in r and denote it by $I_w(r)$. In this case we say that r is *w-finite*. Otherwise, we say that r is *w-infinite*. The index of a regular expression can be used to give a lowerbound on its size according to the following lemma.

Lemma 3.6 ([8]). *For any regular expression r and string w, if r is w-finite, then $I_w(r) < 2|r|$.*[2]

Now, we can state the most important property of K_n.

Lemma 3.7. *Let $n \geq 2$. For any $C \subseteq \{0, \ldots, n-1\}$ of cardinality k and $i \in C$, there exists a string $w \in K_n$ with start- and end-point i only containing integers in C, such that any normal regular expression r which covers w is of size at least 2^k.*

Proof. The proof is by induction on the value of k. For $k = 1$, $C = \{i\}$. Then, define $w = \text{enc}(i)\$\text{enc}(i)\#$, which satisfies all conditions and any expression covering w must definitely have a size of at least 2.

For the inductive step, let $C = \{j_1, \ldots, j_k\}$. Define $C_\ell = C \setminus \{j_{(\ell \mod k)+1}\}$ and let w_ℓ be the string given by the induction hypothesis with respect to C_ℓ (of size $k-1$) and j_ℓ. Note that $j_\ell \in C_\ell$. Further, define $m = 2^{k+1}$ and set

$$w = \text{enc}(j_1)\$\text{enc}(i)\#w_1^m\text{enc}(j_2)\$\text{enc}(j_1)\#w_2^m\text{enc}(j_3)\$\text{enc}(j_2)\#\cdots w_k^m\text{enc}(i)\$\text{enc}(j_k)\#.$$

Then, $w \in K_n$, has i as start and end-point and only contains integers in C. It only remains to show that any expression r which is normal and covers w is of size at least 2^k.

Fix such a regular expression r. If r is w_ℓ-finite for some $\ell \leq k$. Then, $I_{w_\ell}(r_k) \geq m = 2^{k+1}$ by construction of w. By Lemma 3.6, $|r| \geq 2^k$ and we are done.

Therefore, assume that r is w_ℓ-infinite for every $\ell \leq k$. For every $\ell \leq k$, consider all subexpressions of r which are w_ℓ-infinite. It is easy to see that all minimal elements in this set of subexpressions must be starred subexpressions. Here and in the following, we say that an expression is minimal with respect to a set simply when no other expression in the set is a subexpression. Indeed, a subexpression of the form a or ε can never be w_ℓ-infinite and a subexpression of the form $r_1 r_2$ or $r_1 + r_2$ can only be w_ℓ-infinite if r_1 and/or r_2 are w_ℓ-infinite and is thus not minimal with respect to w_ℓ-infinity. Among these minimal starred subexpressions for w_ℓ, choose one and denote it by s_ℓ. Let $E = \{s_1, \ldots, s_k\}$. Note that since r is normal, all its subexpressions are also normal. As in addition each s_ℓ covers w_ℓ, by the induction hypothesis the size of each s_ℓ is at least 2^{k-1}. Now, choose from E some expression s_ℓ such that s_ℓ is minimal with respect to the other elements in E.

As r is normal and s_ℓ is a starred subexpression of r, there is an integer j such that every non-empty string in $L(s_\ell)$ contains j. By definition of the strings $w_1, \ldots, w_k$, there is some w_p, $p \leq k$, such that w_p does not contain j. Denote by s_p the starred subexpression from E which is w_p-infinite. In particular, s_ℓ and s_p cannot be the same subexpression of r.

Now, there are three possibilities:

- s_ℓ and s_p are completely disjoint subexpressions of r. That is, they are both not a subexpression of one another. By induction they must both be of size 2^{k-1} and thus $|r| \geq 2^{k-1} + 2^{k-1} = 2^k$.
- s_p is a strict subexpression of s_ℓ. This is not possible since s_ℓ is chosen to be a minimum element from E.
- s_ℓ is a strict subexpression of s_p. We show that if we replace s_ℓ by ε in s_p, then s_p is still w_p-infinite. It then follows that s_p still covers w_p, and thus s_p without s_ℓ is of size at least 2^{k-1}. As $|s_\ell| \geq 2^{k-1}$ as well it follows that $|r| \geq 2^k$.

[2] In fact, in [8] the length of an expression is defined as the number of Σ-symbols occurring in it. However, since our length measure also contains these Σ-symbols, this lemma still holds in our setting.

To see that s_p without s_ℓ is still w_p-infinite, recall that any non-empty string defined by s_ℓ contains j and j does not occur in w_p. Therefore, a full iteration of s_ℓ can never contribute to the matching of any number of repetitions of w_p. So, s_p can only lose its w_p-infinity by this replacement if s_ℓ contains a subexpression which is itself w_p-infinite. However, this then also is a subexpression of s_p and s_p is chosen to be minimal with respect to w_p-infinity, a contradiction. We can only conclude that s_p without s_ℓ is still w_p-infinite.

■

Since by Lemma 3.5 any expression defining K_n is normal, Theorem 3.4(1) directly follows from Lemma 3.7 by choosing $i = 0$, $k = n$. This concludes the proof of Theorem 3.4(1).

4. Complementing regular expressions

It is known that extended regular expressions are non-elementary more succinct than classical ones [7, 33]. Intuitively, each exponent in the tower requires nesting of an additional complement. In this section, we show that in defining the complement of a single regular expression, a double-exponential size increase cannot be avoided in general. In contrast, when the expression is one-unambiguous its complement can be computed in polynomial time.

Theorem 4.1. *(1) For every regular expression r over Σ, a regular expression s with $L(s) = \Sigma^* \setminus L(r)$ can be constructed in time $\mathcal{O}(2^{|r|+1} \cdot |\Sigma| \cdot 4^{2^{|r|+1}})$.*

(2) Let Σ be a four-letter alphabet. For every $n \in \mathbb{N}$, there is a regular expressions r_n of size $\mathcal{O}(n)$ such that any regular expression r defining $\Sigma^* \setminus L(r_n)$ is of size at least 2^{2^n}.

Proof. (2) Take Σ as Σ_K, that is, $\{0, 1, \$, \#\}$. Let $n \in \mathbb{N}$. We define an expression r_n of size $\mathcal{O}(n)$, such that $\Sigma^* \setminus L(r_n) = K_{2^n}$. By Theorem 3.4, any regular expression defining K_{2^n} is of size exponential in 2^n, that is, of size 2^{2^n}. By $r^{[0,n-1]}$ we abbreviate the expression $(\varepsilon + r(\varepsilon + r(\varepsilon \cdots (\varepsilon + r))))$, with a *nesting depth* of $n-1$. We then define r_n as the disjunction of the following expressions:

- all strings that do not start with a prefix in $(0+1)^n\$$:
$$\Sigma^{[0,n]} + (0+1)^{[0,n-1]}(\$ + \#)\Sigma^* + (0+1)^n(0+1+\#)\Sigma^*$$
- all strings where a \$ is not followed by a string in $(0+1)^n\#$:
$$\Sigma^*\$\big(\Sigma^{[0,n-1]}(\# + \$) + \Sigma^n(0+1+\$)\big)\Sigma^*$$
- all strings where a non-final # is not followed by a string in $(0+1)^n\$$:
$$\Sigma^*\#\big(\Sigma^{[0,n-1]}(\# + \$) + \Sigma^n(0+1+\#)\big)\Sigma^*$$
- all strings that do not end in #:
$$\Sigma^*(0+1+\$)$$
- all strings where the corresponding bits of corresponding blocks are different:
$$((0+1)^* + \Sigma^*\#(0+1)^*)0\Sigma^{3n+2}1\Sigma^* + ((0+1)^* + \Sigma^*\#(0+1)^*)1\Sigma^{3n+2}0\Sigma^*.$$

It should be clear that a string over $\{0, 1, \$, \#\}$ is matched by none of the above expressions if and only if it belongs to K_{2^n}. So, the complement of r_n defines exactly K_{2^n}. ■

The previous theorem essentially shows that in complementing a regular expression, there is no better algorithm than translating to a DFA, computing the complement and translating back to a regular expression which includes two exponential steps. However, when the given regular expression is one-unambiguous, a corresponding DFA can be computed in quadratic time through the Glushkov construction [6] eliminating already one exponential step. The proof of the next theorem shows that the complement of that DFA can be directly defined by a regular expression of polynomial size.

Theorem 4.2. *For any one-unambiguous regular expression r over an alphabet Σ, a regular expression s defining $\Sigma^* \setminus L(r)$ can be constructed in time $\mathcal{O}(n^3)$, where n is the size of r.*

Proof. Let r be a one-unambiguous expression over Σ. We introduce some notation.

- The set Not-First(r) contains all Σ-symbols which are not the first symbol in any word defined by r, that is, Not-First$(r) = \Sigma \setminus \{a \mid a \in \Sigma \wedge \exists w \in \Sigma^*, aw \in L(r)\}$.
- For any symbol $x \in \mathrm{Sym}(r^\flat)$, the set Not-Follow$(r, x)$ contains all Σ-symbols of which no marked version can follow x in any word defined by $r^\flat$. That is, Not-Follow$(r, x) = \Sigma \setminus \{y^\natural \mid y \in \mathrm{Sym}(r^\flat) \wedge \exists w, w' \in \mathrm{Sym}(r^\flat)^*, wxyw' \in L(r^\flat)\}$.
- The set Last(r) contains all *marked* symbols which are the last symbol of some word defined by $r^\flat$. Formally, Last$(r) = \{x \mid x \in \mathrm{Sym}(r^\flat) \wedge \exists w \in \Sigma^*, wx \in L(r^\flat)\}$.

We define the following regular expressions:

- $\mathrm{init}(r) = \begin{cases} \text{Not-First}(r)\Sigma^* & \text{if } \varepsilon \in L(r); \text{ and} \\ \varepsilon + \text{Not-First}(r)\Sigma^* & \text{if } \varepsilon \notin L(r). \end{cases}$
- For every $x \in \mathrm{Sym}(r^\flat)$, let $r_x^\flat$ be the expression defining $\{wx \mid w \in \mathrm{Sym}(r^\flat)^* \wedge \exists u \in \mathrm{Sym}(r^\flat)^*, wxu \in L(r^\flat)\}$. That is, all prefixes of strings in $r^\flat$ ending in x. Then, let r_x define $L(r_x^\flat)^\natural$.

We are now ready to define s:

$$\mathrm{init}(r) \quad + \bigcup_{x \notin \mathrm{Last}(r)} r_x(\varepsilon \quad + \quad \text{Not-Follow}(r, x)\Sigma^*) \quad + \bigcup_{x \in \mathrm{Last}(r)} r_x \text{Not-Follow}(r, x)\Sigma^*.$$

It can be shown that s can be constructed in time cubic in the size of r and that s defines the complement of r. The latter is proved by exhibiting a direct correspondence between s and the complement of the Glushkov automaton of r. ∎

We conclude this section by remarking that one-unambiguous regular expressions are not closed under complement and that the constructed s is therefore not necessarily one-unambiguous.

5. Intersecting regular expressions

In this section, we study the succinctness of intersection. In particular, we show that the intersection of two (or any fixed number) and an arbitrary number of regular expressions are exponentially and double exponentially more succinct than regular expressions, respectively. Actually, the exponential bound for a fixed number of expressions already holds for single-occurrence regular expressions, whereas the double exponential bound for an arbitrary number of expressions only carries over to one-unambiguous expressions. For single-occurrence expressions this can again be done in exponential time.

In this respect, we introduce a slightly altered version of K_n.

Definition 5.1. Let $\Sigma_L = \{0, 1, \$, \#, \triangle\}$. For all $n \in \mathbb{N}$, $L_n = \{\rho_n(w)\triangle \mid w \in Z_n \wedge |w| \text{ is even}\}$.

We also define a variant of Z_n which only slightly alters the $a_{i,j}$ symbols in Z_n. Thereto, let $\Sigma_n^\circ = \{a_{i^\circ,j}, a_{i,j^\circ} \mid 0 \leq i, j < n\}$ and set $\hat{\rho}(a_{i,j}a_{j,k}) = \triangleright_i a_{i,j^\circ}a_{j^\circ,k}$ and $\hat{\rho}(a_{i_0,i_1}a_{i_1,i_2} \cdots a_{i_{k-2},i_{k-1}}a_{i_{k-1},i_k}) = \hat{\rho}(a_{i_0,i_1}a_{i_1,i_2}) \cdots \hat{\rho}(a_{i_{k-2},i_{k-1}}a_{i_{k-1},i_k})$, where k is even.

Definition 5.2. Let $n \in \mathbb{N}$ and $\Sigma_M^n = \Sigma_n^\circ \cup \{\triangleright_0, \triangle_0, \ldots, \triangleright_{n-1}, \triangle_{n-2}\}$. Then, $M_n = \{\hat{\rho}(w)\triangle_i \mid w \in Z_n \wedge |w| \text{ is even} \wedge i \text{ is the end-point of } w\}$.

Note that paths in M_n are those in Z_n where every odd position is promoted to a circled one ($^\circ$), and triangles labeled with the non-circled positions are added. For instance, the string $a_{2,4}a_{4,3}a_{3,3}a_{3,0} \in Z_5$ is mapped to the string $\triangleright_2 a_{2,4^\circ}a_{4^\circ,3} \triangleright_3 a_{3,3^\circ}a_{3^\circ,0}\triangle_0 \in M_5$.

We make use of the following property:

Lemma 5.3. *Let $n \in \mathbb{N}$.*

(1) Any regular expression defining L_n is of size at least 2^n.
(2) Any regular expression defining M_n is of size at least 2^{n-1}.

The next theorem shows the succinctness of the intersection operator.

Theorem 5.4. *(1) For any $k \in \mathbb{N}$ and regular expressions $r_1, \ldots, r_k$, a regular expression defining $\bigcap_{i \leq k} L(r_k)$ can be constructed in time $\mathcal{O}((m+1)^k \cdot |\Sigma| \cdot 4^{(m+1)^k})$, where $m = \max\{|r_i| \mid 1 \leq i \leq k\}$.*
(2) For every $n \in \mathbb{N}$, there are SOREs r_n and s_n of size $\mathcal{O}(n^2)$ such that any regular expression defining $L(r_n) \cap L(s_n)$ is of size at least 2^{n-1}.
(3) For each $r \in RE(\cap)$ an equivalent regular expression can be constructed in time $\mathcal{O}(2^{|r|} \cdot |\Sigma| \cdot 4^{2^{|r|}})$.
(4) For every $n \in \mathbb{N}$, there are one-unambiguous regular expressions $r_1, \ldots, r_m$, with $m = 2n+1$, of size $\mathcal{O}(n)$ such that any regular expression defining $\bigcap_{i \leq m} L(r_i)$ is of size at least 2^{2^n}.
(5) Let $r_1, \ldots, r_n$ be SOREs. A regular expression defining $\bigcap_{i \leq n} L(r_n)$ can be constructed in time $\mathcal{O}(m \cdot |\Sigma| \cdot 4^m)$, where $m = \sum_{i \leq n} |r_i|$.

Proof. (2) Let $n \in \mathbb{N}$. By Lemma 5.3(2), any regular expression defining M_n is of size at least 2^{n-1}. We define SOREs r_n and s_n of size quadratic in n, such that $L(r_n) \cap L(s_n) = M_n$. We start by partitioning Σ_M^n in two different ways. To this end, for every $i < n$, define $\text{Out}_i = \{a_{i,j^\circ} \mid 0 \leq j < n\}$, $\text{In}_i = \{a_{j^\circ,i} \mid 0 \leq j < n\}$, $\text{Out}_{i^\circ} = \{a_{i^\circ,j} \mid 0 \leq j < n\}$, and, $\text{In}_{i^\circ} = \{a_{j,i^\circ} \mid 0 \leq j < n\}$. Then,

$$\Sigma_M^n = \bigcup_i \text{In}_i \cup \text{Out}_i \cup \{\triangleright_i, \triangle_i\} = \bigcup_{i^\circ} \text{In}_{i^\circ} \cup \text{Out}_{i^\circ} \cup \{\triangleright_i, \triangle_i\}.$$

Further, define

$$r_n = ((\triangleright_0 + \cdots + \triangleright_{n-1}) \bigcup_{i^\circ} \text{In}_{i^\circ}\text{Out}_{i^\circ})^+ (\triangle_0 + \cdots + \triangle_{n-1})$$

and

$$s_n = \Big(\bigcup_i (\text{In}_i + \varepsilon)(\triangleright_i + \triangle_i)(\text{Out}_i + \varepsilon)\Big)^*.$$

Now, r_n checks that every string consists of a sequence of blocks of the form $\triangleright_i a_{j,k^\circ}a_{k^\circ,\ell}$, for $i, j, k, \ell < n$, ending with a $\triangle_i$, for $i < n$. It thus sets the format of the strings and

checks whether the circled indices are equal. Further, s_n checks whether the non-circled indices are equal and whether the triangles have the correct indices. Since the alphabet of M_n is of size $\mathcal{O}(n^2)$, also r_n and s_n are of size $\mathcal{O}(n^2)$.
(4) Let $n \in \mathbb{N}$. We define $m = 2n + 1$ one-unambiguous regular expressions of size $\mathcal{O}(n)$, such that their intersection defines L_{2^n}. By Lemma 5.3(1), any regular expression defining L_{2^n} is of size at least 2^{2^n} and the theorem follows. For ease of readability, we denote Σ_L simply by Σ. The expressions are as follows. There should be an even length sequence of blocks:

$$\big((0+1)^n\$(0+1)^n\#(0+1)^n\$(0+1)^n\#\big)^*\triangle.$$

For all $i \in \{0, \ldots, n-1\}$, the $(i+1)$th bit of the two numbers surrounding an odd # should be equal:

$$\big(\Sigma^i(0\Sigma^{3n+2}0 + 1\Sigma^{3n+2}1)\Sigma^{n-i-1}\#\big)^*\triangle.$$

For all $i \in \{0, \ldots, n-1\}$, the $(i+1)$th bit of the two numbers surrounding an even # should be equal:

$$\Sigma^{2n+2}\Big(\Sigma^i(0\Sigma^{2n-i+1}(\triangle + \Sigma^{n+i+1}0\Sigma^{n-i-1}\#) + (1\Sigma^{2n-i+1}(\triangle + \Sigma^{n+i+1}1\Sigma^{n-i-1}\#)))\Big)^*.$$

Clearly, the intersection of the above expressions defines L_{2^n}. Furthermore, every expression is of size $\mathcal{O}(n)$ and is one-unambiguous as the Glushkov construction translates them into a DFA [6]. ∎

6. Conclusion

In this paper we showed that the complement and intersection of regular expressions are double exponentially more succinct than ordinary regular expressions. For complement, complexity can be reduced to polynomial for the class of one-unambiguous regular expressions although the obtained expressions could fall outside that class. For intersection, restriction to SOREs reduces complexity to exponential. It remains open whether there are natural classes of regular expressions for which both the complement and intersection can be computed in polynomial time.

Acknowledgment. We thank Juraj Hromkovič for sending us reference [35].

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. AW, 1974.
[2] G.J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *Very Large Data Bases*, pp. 115-126, 2006.
[3] G.J. Bex, F. Neven, and Stijn Vansummeren. Inferring XML Schema Definitions from XML data. In *Very Large Data Bases*, pp. 998-1009, 2007.
[4] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML). World Wide Web Consortium, 2004. `http://www.w3.org/TR/REC-xml/`.
[5] A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197-213, 1993.
[6] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182-206, 1998.
[7] Z. R. Dang. On the complexity of a finite automaton corresponding to a generalized regular expression. *Dokl. Akad. Nauk SSSR*, 1973.
[8] A. Ehrenfeucht and H. Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134-146, 1976.

[9] K. Ellul, B. Krawetz, J. Shallit, and M. Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407-437, 2005.
[10] M. Fürer. The complexity of the inequivalence problem for regular expressions with intersection. In *International Colloquium on Automata, Languages and Programming*, pp. 234-245, 1980.
[11] W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for XML: Numerical constraints and interleaving. In *International Conference on Database Theory*, pp. 269-283, 2007.
[12] W. Gelade and F. Neven. Succinctness of pattern-based schema languages for XML. In *Database Programming Languages* 2007, LNCS 4797, pp. 202-216
[13] G. Ghelli, D. Colazzo, and C. Sartiani. Efficient inclusion for a class of XML types with interleaving and counting. In *Database Programming Languages* 2007, LNCS 4797, pp. 231-245.
[14] N. Globerman and D. Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 169(2):161-184, 1996.
[15] M. Grohe and N. Schweikardt. The succinctness of first-order logic on linear orders. *Logical Methods in Computer Science*, 1(1), 2005.
[16] H. B. Hunt III. The equivalence problem for regular expressions with intersection is not polynomial in tape. Technical report, Department of Computer Science, Cornell University, 1973.
[17] L. Ilie and S. Yu. Algorithms for computing small nfas. In *Mathematical Foundations of Computer Science*, pp. 328-340, 2002.
[18] T. Jiang and B. Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40(1):25-31, 1991.
[19] D. Kozen. Lower bounds for natural proof systems. In *FOCS 1977*, pp. 254-266, IEEE.
[20] O. Kupferman and S. Zuhovitzky. An improved algorithm for the membership problem for extended regular expressions. In *Mathematical Foundations of Computer Science*, pp. 446-458, 2002.
[21] L. Libkin. Logics for unranked trees: An overview. In *International Colloquium on Automata, Languages and Programming*, pp. 35-50, 2005.
[22] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions. In *Mathematical Foundations of Computer Science*, pp. 889-900, 2004.
[23] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770-813, 2006.
[24] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9(1):39-47, 1960.
[25] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technologies*, 5(4):660-704, 2005.
[26] G. Myers. A four Russians algorithm for regular pattern matching. *J. of the ACM*, 39(2):432-448, 1992.
[27] F. Neven. Automata, logic, and XML. In *Conference for Computer Science Logic*, pp. 2-26, 2002.
[28] H. Petersen. The membership problem for regular expressions with intersection is complete in LOGCFL. In *Proc. STACS 2002*, Lect. Notes in Comp. Science **2285**, pp. 513-522, Springer (2002).
[29] J. M. Robson. The emptiness of complement problem for semi extended regular expressions requires c^n space. *Information Processing Letters*, 9(5):220-222, 1979.
[30] G. Rosu and M. Viswanathan. Testing extended regular language membership incrementally by rewriting. In *Rewriting Techniques and Applications*, pp. 499-514, 2003.
[31] T. Schwentick. Automata for XML - a survey. *J. Comp. and System Sciences*, 73(3), 289-315, 2007.
[32] C.M. Sperberg-McQueen and H. Thompson. XML Schema. `http://www.w3.org/XML/Schema,2005`.
[33] L. Stockmeyer and A. Meyer. Word problems requiring exponential time. In *Symposium on the Theory of Computing*, pp. 1-9, 1973.
[34] V. Vianu. Logic as a query language: From Frege to XML. In *Proc. STACS 2003*, pp. 1-12, Lect. Notes in Comp. Science **2607**, Springer (2003).
[35] V. Waizenegger. Uber die Effizienz der Darstellung durch reguläre Ausdrücke und endliche Automaten. Diplomarbeit, RWTH Aachen, 2000.
[36] S. Yu. Handbook of formal languages. volume 1, chapter 2, pp. 41-110. Springer, 1997.
[37] D. Ziadi. Regular expression for a language without empty word. *Theoretical Computer Science*, 163(1&2):309-315, 1996.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 337-348
www.stacs-conf.org

EFFICIENT ALGORITHMS FOR MEMBERSHIP IN BOOLEAN HIERARCHIES OF REGULAR LANGUAGES

C. GLASSER [1], H. SCHMITZ [2], AND V. SELIVANOV [3]

[1] Universität Würzburg, Germany.
E-mail address: glasser@informatik.uni-wuerzburg.de

[2] Fachhochschule Trier, Germany.
E-mail address: schmitz@informatik.fh-trier.de

[3] A. P. Ershov Institute of Informatics Systems, Russia.
E-mail address: vseliv@nspu.ru

Abstract. The purpose of this paper is to provide *efficient* algorithms that decide membership for classes of several Boolean hierarchies for which efficiency (or even decidability) were previously not known. We develop new forbidden-chain characterizations for the single levels of these hierarchies and obtain the following results:

- The classes of the Boolean hierarchy over level Σ_1 of the dot-depth hierarchy are decidable in NL (previously only the decidability was known). The same remains true if predicates mod d for fixed d are allowed.
- If modular predicates for arbitrary d are allowed, then the classes of the Boolean hierarchy over level Σ_1 are decidable.
- For the restricted case of a two-letter alphabet, the classes of the Boolean hierarchy over level Σ_2 of the Straubing-Thérien hierarchy are decidable in NL. This is the first decidability result for this hierarchy.
- The membership problems for all mentioned Boolean-hierarchy classes are logspace many-one hard for NL.
- The membership problems for quasi-aperiodic languages and for d-quasi-aperiodic languages are logspace many-one complete for PSPACE.

Introduction

The study of decidability and complexity questions for classes of regular languages is a central research topic in automata theory. Its importance stems from the fact that finite automata are fundamental to many branches of computer science, e.g., databases, operating systems, verification, hardware and software design.

Key words and phrases: automata and formal languages, computational complexity, dot-depth hierarchy, Boolean hierarchy, decidability, efficient algorithms.

This work was done during a stay of the third author at the University of Würzburg, supported by DFG Mercator program and by RFBR grant 07-01-00543a.

There are many examples for decidable classes of regular languages (e.g., locally testable languages), while the decidability of other classes is still a challenging open question (e.g., dot-depth two, generalized star-height). Moreover, among the decidable classes there is a broad range of complexity results. For some of them, e.g., the class of piecewise testable languages, efficient algorithms are known that work in nondeterministic logarithmic space (NL) and hence in polynomial time. For other classes, a membership test needs more resources, e.g., deciding the membership in the class of star-free languages is PSPACE-complete.

The purpose of this paper is to provide *efficient* algorithms that decide membership for classes of several Boolean hierarchies for which efficiency (or even decidability) were not previously known. Many of the known efficient decidability results for classes of regular languages are based on so-called forbidden-pattern characterizations. Here a language belongs to a class of regular languages if and only if its deterministic finite automaton does *not* have a certain subgraph (the forbidden pattern) in its transition graph. Usually, such a condition can be checked efficiently, e.g., in nondeterministic logarithmic space [Ste85a, CPP93, GS00a, GS00b].

However, for the Boolean hierarchies considered in this paper, the design of efficient algorithm is more involved, since here no forbidden-pattern characterizations are known. More precisely, wherever decidability is known, it is obtained from a characterization of the corresponding class in terms of *forbidden* alternating chains of word extensions. Though the latter also is a forbidden property, the known characterizations are not efficiently checkable in general. (Exceptions are the special 'local' cases $\Sigma_1^{\varrho}(n)$ and $\mathcal{C}_k^1(n)$ where decidability in NL is known [SW98, Sch01].) To overcome these difficulties, we first develop alternative forbidden-chain characterizations (they essentially ask only for certain reachability conditions in transition graphs). From our new characterizations we obtain efficient algorithms for membership tests in NL. For two of the considered Boolean hierarchies, these are the first decidable characterizations at all, i.e., for the classes $\Sigma_2^{\varrho}(n)$ for the alphabet $A = \{a, b\}$, and for the classes $\Sigma_1^{\tau}(n)$).

Definitions. We sketch the definitions of the Boolean hierarchies considered in this paper. Σ_1^{ϱ} denotes the class of languages definable by first-order Σ_1-sentences over the signature $\varrho = \{\leq, Q_a, \ldots\}$ where for every letter $a \in A$, $Q_a(i)$ is true if and only if the letter a appears at the i-th position in the word. Σ_1^{ϱ} equals level 1/2 of the Straubing-Thérien hierarchy (STH for short) [Str81, Thé81, Str85, PP86]. Σ_2^{ϱ} is the class of languages definable by similar first-order Σ_2-sentences; this class equals level 3/2 of the Straubing-Thérien hierarchy. Let σ be the signature obtained from ϱ by adding constants for the minimum and maximum positions in words and adding functions that compute the successor and the predecessor of positions. Σ_1^{σ} denotes the class of languages definable by first-order Σ_1-sentences of the signature σ; this class equals level 1/2 of the dot-depth hierarchy (DDH for short) [CB71, Tho82]. Let τ_d be the signature obtained from σ by adding the unary predicates $P_d^0, \ldots, P_d^{d-1}$ where $P_d^j(i)$ is true if and only if $i \equiv j \pmod d$. Let τ be the union of all τ_d. $\Sigma_1^{\tau_{\mathrm{d}}}$ (resp., Σ_1^{τ}) is the class of languages definable by first-order Σ_1-sentences of the signature τ_d (resp., τ). $\mathcal{C}_k^d$ is the generalization of Σ_1^{ϱ} where neighborhoods of $k+1$ consecutive letters and distances modulo d are expressible (Definition 1.2). For a class $\mathcal{D}$ (in our case one of the classes Σ_1^{ϱ}, Σ_1^{σ}, $\mathcal{C}_k^d$, $\Sigma_1^{\tau_{\mathrm{d}}}$, Σ_1^{τ}, and Σ_2^{ϱ} for $|A| = 2$), the *Boolean hierarchy over* $\mathcal{D}$ is the family of classes

$$\mathcal{D}(n) \stackrel{\mathrm{df}}{=} \{L \mid L = L_1 - (L_2 - (\ldots - L_n)) \text{ where } L_1, \ldots, L_n \in \mathcal{D} \text{ and } L_1 \supseteq L_2 \supseteq \cdots \supseteq L_n\}.$$

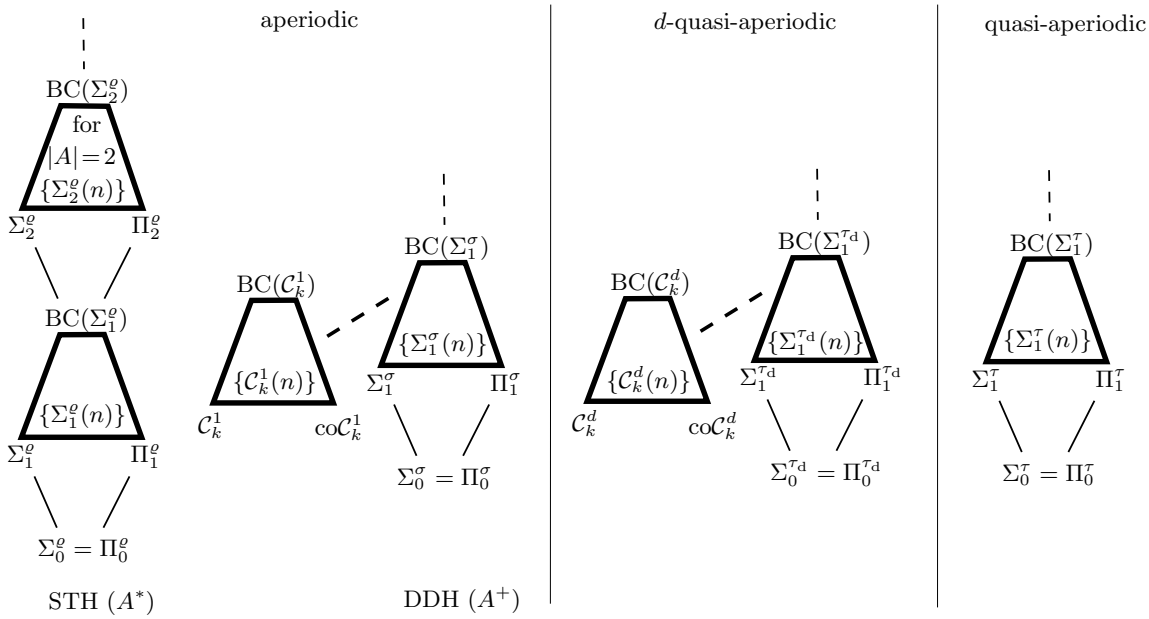


Figure 1: Boolean hierarchies considered in this paper.

The Boolean hierarchies considered in this paper are illustrated in Figure 1.

Our Contribution. The paper contributes to the understanding of Boolean hierarchies of regular languages in two ways:

(1) For the classes $\Sigma_1^{\sigma}(n)$, $\Sigma_1^{\tau_{\mathrm{d}}}(n)$, and $\Sigma_2^{\varrho}(n)$ for the alphabet $A = \{a, b\}$ we prove new characterizations in terms of forbidden alternating chains. In case of $\Sigma_2^{\varrho}(n)$ for the alphabet $A = \{a, b\}$, this is the first characterization of this class.

(2) For the classes $\Sigma_1^{\sigma}(n)$, $\mathcal{C}_k^d(n)$, $\Sigma_1^{\tau_{\mathrm{d}}}(n)$, and $\Sigma_2^{\varrho}(n)$ for the alphabet $A = \{a, b\}$ we construct the first efficient algorithms for testing membership in these classes. In particular, this yields the decidability of the classes $\Sigma_1^{\tau}(n)$, and of $\Sigma_2^{\varrho}(n)$ for the alphabet $A = \{a, b\}$.

We also show that the membership problems for all mentioned Boolean-hierarchy classes are logspace many-one hard for NL. An overview of the obtained decidability and complexity results can be found in Table 1. Moreover, we prove that the membership problems for quasi-aperiodic languages and for d-quasi-aperiodic languages are logspace many-one complete for PSPACE.

Boolean hierarchies can also be seen as fine-grain measures for regular languages in terms of descriptional complexity. Note that the Boolean hierarchies considered in this paper do not collapse [Shu98, SS00, Sel04]. Moreover, all these hierarchies either are known or turn out to be decidable (see Table 1 for the attribution of these results). If in addition the Boolean closure of the base class is decidable, then we can even exactly compute the Boolean level of a given language. By known results (summarized in Theorem 1.1), one can do this exact computation of the level for the Boolean hierarchies over Σ_1^{ϱ}, Σ_2^{ϱ} (for alphabet $A = \{a, b\}$), $\mathcal{C}_k^1$, Σ_1^{σ}, and Σ_1^{τ}. To achieve the same for the Boolean hierarchies over $\mathcal{C}_k^d$ and $\Sigma_1^{\tau_{\mathrm{d}}}$ we need the decidability of their Boolean closures which is not known.

Related Work. Due to the many characterizations of regular languages there are several approaches to attack decision problems on subclasses of regular languages: Among

Boolean hierarchy classes	decidability	complexity
$\Sigma_1^{\varrho}(n)$	[SW98]	NL-complete [SW98]
$\mathcal{C}_k^1(n)$	[GS01a, Sel01]	NL-complete [Sch01]
$\Sigma_1^{\sigma}(n)$	[GS01a]	NL-complete [this paper]
$\mathcal{C}_k^d(n)$	[Sel04]	NL-complete [this paper]
$\Sigma_1^{\tau_{\mathrm{d}}}(n)$	[Sel04]	NL-complete [this paper]
$\Sigma_1^{\tau}(n)$	[this paper]	no efficient bound known (see Remark 4.3)
$\Sigma_2^{\varrho}(n)$ for $\lvert A\rvert = 2$	[this paper]	NL-complete [this paper]

Table 1: Overview of decidability and complexity results.

them there is the algebraic, the automata-theoretic, and the logical approach. In this paper we mainly use the logical approach which has a long tradition starting with the early work of Trakhtenbrot [Tra58] and Büchi [Büc60]. Decidability questions for Boolean hierarchies over classes of concatenation hierarchies were previously studied by [SW98, Sch01, GS01a, Sel04]. Enrichments of the first-order logics related to the dot-depth hierarchy and the Straubing-Thérien hierarchy were considered in [BCST92, Str94, MPT00, Sel04, CPS06]. For more background on regular languages, starfree languages, concatenation hierarchies, and their decidability questions we refer to the survey articles [Brz76, Pin95, Pin96a, Pin96b, Yu96, PW02, Wei04].

Paper Outline. After the preliminaries, we explain the general idea of an efficient membership algorithm for the classes $\mathcal{C}_k^d(n)$ (section 2). This easy example shows how a suitable characterization of a Boolean hierarchy can be turned into an efficient membership test. The algorithms for the other Boolean hierarchies are similar, but more complicated. Section 3 provides new alternating-chain characterizations for the Boolean hierarchies over Σ_1^{σ}, $\Sigma_1^{\tau_{\mathrm{d}}}$, and Σ_2^{ϱ} for the alphabet $A = \{a, b\}$. In section 4 we exploit these characterizations and obtain efficient algorithms for testing the membership in these classes. In particular, we obtain the decidability of the classes $\Sigma_1^{\tau}(n)$ and $\Sigma_2^{\varrho}(n)$ for the alphabet $A = \{a, b\}$. Finally, section 5 provides lower bounds for the complexity of the considered decidability problems. As a consequence (with the exception of $\Sigma_1^{\tau}(n)$) the membership problems of all considered Boolean levels are logspace many-one complete for NL. In contrast, the membership problems of the general classes FO_{τ} and FO_{τ_d} are logspace many-one complete for PSPACE and hence are strictly more complex.

Detailed proofs are available in the technical report [GSS07].

1. Preliminaries

In this section we recall definitions and results that are needed later in the paper. If not stated otherwise, A denotes some finite alphabet with $|A| \geq 2$. Let A^* and A^+ be the sets of finite (resp., of finite non-empty) words over A. If not stated otherwise, variables range over the set of natural numbers. We use $[m, n]$ as abbreviation for the interval $\{m, m+1, \ldots, n\}$. For a deterministic finite automaton $M = (A, Z, \delta, s_0, F)$ (dfa for short), the number of states is denoted by $|M|$ and the accepted language is denoted by $L(M)$. Moreover, for words x and y we write $x \equiv_M y$ if and only if $\delta(s_0, x) = \delta(s_0, y)$. For a

class of languages $\mathcal{C}$, BC($\mathcal{C}$) denotes the Boolean closure of $\mathcal{C}$, i.e., the closure under union, intersection, and complementation.

All hardness and completeness results in this paper are with respect to logspace many-one reductions, i.e., whenever we refer to NL-complete sets (resp., PSPACE-complete sets) then we mean sets that are logspace many-one complete for NL (resp., PSPACE).

1.1. The Logical Approach to Regular Languages

We relate to an arbitrary alphabet $A = \{a, \ldots\}$ the signatures $\varrho = \{\leq, Q_a, \ldots\}$ and $\sigma = \{\leq, Q_a, \ldots, \bot, \top, p, s\}$, where $\leq$ is a binary relation symbol, Q_a (for any $a \in A$) is a unary relation symbol, $\bot$ and $\top$ are constant symbols, and p, s are unary function symbols. A word $u = u_0 \ldots u_n \in A^+$ may be considered as a structure $\mathbf{u} = (\{0, \ldots, n\}; \leq, Q_a, \ldots)$ of signature σ, where $\leq$ has its usual meaning, $Q_a (a \in A)$ are unary predicates on $\{0, \ldots, n\}$ defined by $Q_a(i) \Leftrightarrow u_i = a$, the symbols $\bot$ and $\top$ denote the least and the greatest elements, while p and s are respectively the predecessor and successor functions on $\{0, \ldots, n\}$ (with $p(0) = 0$ and $s(n) = n$). Similarly, a word $v = v_1 \ldots v_n \in A^*$ may be considered as a structure $\mathbf{v} = (\{1, \ldots, n\}; \leq, Q_a, \ldots)$ of signature ϱ. For a sentence ϕ of σ (resp., ϱ), let $L_\phi = \{u \in A^+ | \mathbf{u} \models \phi\}$ (resp., $L_\phi = \{v \in A^* | \mathbf{v} \models \phi\}$). Sentences ϕ, ψ are treated as equivalent when $L_\phi = L_\psi$. A language is FO_σ-definable (resp., FO_ϱ-definable) if it is of the form L_ϕ, where ϕ ranges over first-order sentences of σ (resp., ϱ). We denote by Σ_k^σ (resp., Π_k^σ) the class of languages that can be defined by a sentence of σ having at most $k-1$ quantifier alternations, starting with an existential (resp., universal) quantifier. Σ_k^ϱ and Π_k^ϱ are defined analogously.

It is well-known that the class of FO_σ-definable languages (and FO_ϱ-definable languages) coincides with the class of *regular aperiodic languages* which are also known as the *star-free languages*. Moreover there is a levelwise correspondence to concatenation hierarchies: The classes Σ_k^ϱ, Π_k^ϱ, and BC(Σ_k^ϱ) coincide with the classes of the Straubing-Thérien hierarchy [PP86], while the classes Σ_k^σ, Π_k^σ, and BC(Σ_k^σ) coincide with the classes of the dot-depth hierarchy [Tho82].

We will consider also some enrichments of the signature σ. Namely, for any positive integer d let τ_d be the signature $\sigma \cup \{P_d^0, \ldots, P_d^{d-1}\}$, where P_d^r is the unary predicate true on the positions of a word which are equivalent to r modulo d. By FO_{τ_d}-definable language we mean any language of the form L_ϕ, where ϕ is a first-order sentence of signature τ_d. Note that signature τ_1 is essentially the same as σ because P_1^0 is the valid predicate. In contrast, for $d > 1$ the FO_{τ_d}-definable languages need not to be aperiodic. E.g., the sentence $P_2^1(\top)$ defines the language L consisting of all words of even length which is known to be non-aperiodic. We are also interested in the signature $\tau = \bigcup_d \tau_d$. Barrington et al. [BCST92, Str94] defined quasi-aperiodic languages and showed that this class coincides with the class of FO_τ-definable languages. With the same proof we obtain the equality of the class of d-quasi-aperiodic languages and the class of FO_{τ_d}-definable languages [Sel04]. It was observed in the same paper that $\Sigma_n^\tau = \bigcup_d \Sigma_n^{\tau_\mathrm{d}}$ for each $n > 0$, where Σ_n with an upper index denotes the class of regular languages defined by Σ_n-sentences of the corresponding signature in the upper index.

Theorem 1.1. *For the following classes $\mathcal{D}$ it is decidable whether a given dfa M accepts a language in $\mathcal{D}$:* BC(Σ_1^ϱ) [Sim75], BC(Σ_2^ϱ) *for* $|A| = 2$ [Str88], BC(Σ_1^σ) [Kna83], BC(Σ_1^τ) [MPT00].

We do not know the decidability of $\mathrm{BC}(\Sigma_1^{\tau_\mathrm{d}})$. However, it is likely to be a generalization of Knast's proof [Kna83].

1.2. Preliminaries on the Classes $\mathcal{C}_k^d(n)$

We will also refer to 'local' versions of the BH's over Σ_1^σ and $\Sigma_1^{\tau_\mathrm{d}}$ [Ste85a, GS01a, Sel01, Sel04]. For any $k \geq 0$ the following partial order on Σ^+ was studied in [Ste85a, GS01a, Sel01]: $u \leq_k v$, if $u = v \in A^{\leq k}$ or $u, v \in A^{>k}$, $p_k(u) = p_k(v)$, $s_k(u) = s_k(v)$, and there is a k-embedding $f : u \to v$. Here $p_k(u)$ (resp., $s_k(u)$) is the prefix (resp., suffix) of u of length k, and the k-embedding f is a monotone injective function from $\{0\ldots, |u|-1\}$ to $\{0\ldots, |v|-1\}$ such that $u(i)\cdots u(i+k) = v(f(i))\cdots v(f(i)+k)$ for all $i < |u| - k$. Note that $\leq_0$ is the subword relation.

Definition 1.2 ([Sel04])**.** Let $k \geq 0$ and $d > 0$.

(1) We say that a k-embedding $f : u \to v$ is a (k,d)-embedding, if $P_d^r(i)$ implies $P_d^r(f(i))$ for all $i < |u|$ and $r < d$.
(2) For all $u, v \in A^+$, let $u \leq_k^d v$ mean that $u = v \in A^{\leq k}$ or $u, v \in A^{>k}$, $p_k(u) = p_k(v)$, $s_k(u) = s_k(v)$, and there is a (k,d)-embedding $f : u \to v$.
(3) With $\mathcal{C}_k^d$ we denote the class of all upper sets in $(A^+; \leq_k^d)$.

Note that for $d = 1$ the order $\leq_k^d$ coincides with $\leq_k$. By an *alternating $\leq_k^d$-chain* of length n for a set L we mean a sequence $(x_0, \ldots, x_n)$ such that $x_0 \leq_k^d \cdots \leq_k^d x_n$ and $x_i \in L \Leftrightarrow x_{i+1} \notin L$ for every $i < n$. The chain is called *1-alternating* if $x_0 \in L$, otherwise it is called *0-alternating*.

Proposition 1.3 ([GS01a, Sel01, Sel04])**.** *For all $L \subseteq A^+$ and $n \geq 1$, $L \in \mathcal{C}_k^d(n)$ if and only if L has no 1-alternating chain of length n in $(A^+; \leq_k^d)$.*

Moreover, $(A^+; \leq_k^d)$ is a well partial order, $\Sigma_1^{\tau_\mathrm{d}} = \bigcup_k \mathcal{C}_k^d$, and $\Sigma_1^\tau = \bigcup_{k,d} \mathcal{C}_k^d$ [GS01a, Sel01, Sel04].

Theorem 1.4 ([Ste85a])**.** *It is decidable whether a given dfa accepts a language in* $\mathrm{BC}(\mathcal{C}_k^1)$.

For $d > 1$ it is not known whether $\mathrm{BC}(\mathcal{C}_k^d)$ is decidable. However, we expect that this can be shown by generalizing the proof in [Ste85a].

2. Efficient Algorithms for $\mathcal{C}_k^d(n)$

The main objective of this paper is the design of *efficient* algorithms deciding membership for particular Boolean hierarchies. For this, two things are needed: first, we need to prove suitable characterizations for the single levels of these hierarchies. This gives us certain criteria that can be used for testing membership. Second, we need to construct algorithms that efficiently apply these criteria. If both steps are successful, then we obtain an efficient membership test.

Based on known ideas for membership tests for $\mathcal{C}_0^1(n)$ [SW98][1] and $\mathcal{C}_k^1(n)$ [Sch01], in this section we explain the construction of a nondeterministic, logarithmic-space membership algorithm for the classes $\mathcal{C}_k^d(n)$. This is the first efficient membership test for this general

[1] For all n, the classes $\mathcal{C}_0^1(n)$ and $\Sigma_1^\varrho(n)$ coincide up to the empty word, i.e., $\mathcal{C}_0^1(n) = \{L \cap A^+ \mid L \in \Sigma_1^\varrho(n)\}$.

case. Our explanation has an exemplary character, since it shows how a suitable characterization of a Boolean hierarchy can be turned into an efficient membership test. Our results in later sections use similar, but more complicated constructions.

We start with the easiest case $k = 0$ and $d = 1$, i.e., with the classes $\mathcal{C}_0^1(n)$. By Proposition 1.3,

$$L \notin \mathcal{C}_0^1(n) \quad \Leftrightarrow \quad L \text{ has a 1-alternating } \leq_0\text{-chain of length } n. \tag{2.1}$$

We argue that for a given L, represented by a finite automaton M, the condition on the right-hand side can be verified in nondeterministic logarithmic space. So we have to test whether there exists a chain $w_0 \leq_0 \cdots \leq_0 w_n$ such that $w_i \in L$ if and only if i is even. This is done by the following algorithm.

```
0  // On input of a deterministic, finite automaton M = (A, Z, δ, z_0, F)
      the algorithm tests whether L(M) ∈ C_0^1(n).
1  let s_0 = ··· = s_n = z_0
2  do
3     nondeterministically choose a ∈ A and j ∈ [0, n]
4     for i = j to n
5        s_i = δ(s_i, a)        // stands for the imaginary command w_i := w_i a
6     next i
7  until ∀i, [s_i ∈ F ⇔ i is even]
8  accept
```

The algorithm guesses the words $w_0, \ldots, w_n$ in parallel. However, instead of constructing these words in the memory, it guesses the words letter by letter and stores only the states $s_i = \delta(z_0, w_i)$. More precisely, in each pass of the loop we choose a letter a and a number j, and we interpret this choice as appending a to the words $w_j, \ldots, w_n$. Simultaneously, we update the states $s_j, \ldots, s_n$ appropriately. By doing so, we guess all possible chains $w_0 \leq_0 \cdots \leq_0 w_n$ in such a way that we know the states $s_i = \delta(z_0, w_i)$. This allows us to easily verify the right-hand side of (2.1) in line 7. Hence, testing non-membership in $\mathcal{C}_0^1(n)$ is in NL. By NL = coNL [Imm88, Sze87], the membership test also belongs to NL.

The algorithm can be modified such that it works for $\mathcal{C}_0^d(n)$ where d is arbitrary: For this we have to make sure that the guessed $\leq_0$-chain is even a $\leq_0^d$-chain, i.e., the word extensions must be such that the lengths of single insertions are divisible by d. This is done by (i) introducing new variables l_i that count the current length of w_i modulo d and (ii) by making sure that $l_i = l_{i+1}$ whenever $j \leq i < n$ (i.e., letters that appear in both words, w_i and w_{i+1}, must appear at equivalent positions modulo d). So also the membership test for $\mathcal{C}_0^d(n)$ belongs to NL.

Finally, we adapt the algorithm to make it work for $\mathcal{C}_k^d(n)$ where d and k are arbitrary. So we have to make sure that the guessed $\leq_0^d$-chain is even a $\leq_k^d$-chain. For this, let us consider an extension $u \leq_k^d w$ where $u, w \in A^{>k}$. The (k,d)-embedding f that is used in the definition of $u \leq_k^d w$ ensures that for all i it holds that in u at position i there are the same $k+1$ letters as in w at position $f(i)$. Therefore, a word extension $u \leq_k^d w$ can be split into a series of elementary extensions of the form $u_1 u_2 \leq_0^d u_1 v u_2$ such that the length k prefixes of u_2 and vu_2 are equal. The latter is called the *prefix condition*. Moreover, we can always make sure that the positions in u at which the elementary extensions occur form a strictly increasing sequence. This allows us to guess the words in the $\leq_k^d$-chain letter by letter. Now the algorithm can test the prefix condition by introducing new variables v_i that contain a guessed preview of the next k letters in w_i. Each time a letter is appended to

w_i, (i) we verify that this letter is consistent with the preview v_i and (ii) we update v_i by removing the first letter and by appending a new guessed letter. In this way the modified algorithm carries the length k previews of the w_i with it and it makes sure that guessed letters are consistent with these previews. Moreover, we modify the algorithm such that whenever $j \leq i < n$, then the condition $v_i = v_{i+1}$ is tested. The latter makes sure that elementary extensions $u_1u_2 \leq_0^d u_1vu_2$ satisfy the prefix condition and hence the involved words are even in $\leq_k^d$ relation. This modified algorithm shows the following.

Theorem 2.1. $\{M \mid M$ *is a det. finite automaton and* $L(M) \in \mathcal{C}_k^d(n)\} \in$ NL *for* $k \geq 0$, $d \geq 1$.

We now explain why the above idea does not immediately lead to a nondeterministic, logarithmic-space membership algorithm for the classes $\Sigma_1^\sigma(n)$, although an alternating chain characterization for $\Sigma_1^\sigma(n)$ is known from [GS01a]. Note that the described algorithm for $\mathcal{C}_k^d(n)$ stores the following types of variables in logarithmic space.

(1) variables s_i that contain states of M
(2) variables l_i that contain numbers from $[0, d-1]$
(3) variables v_i that contain words of length k

However, the characterization of the classes $\Sigma_1^\sigma(n)$ [GS01a] is unsuitable for our algorithm: In order to verify the forbidden-chain condition, we have to guess a chain of so-called structured words and have to make sure that certain parts u in these words are M-idempotent (i.e., $\delta(s,u) = \delta(s,uu)$ for all states s). Again we would try to guess the words letter by letter, but now we have to make sure that (larger) parts u of these words are M-idempotent. We do not know how to verify the latter condition in logarithmic space.

In a similar way one observes that the known characterization of the classes $\Sigma_1^{\tau_d}(n)$ [Sel04] cannot be used for the construction of an efficient membership test. So new characterizations of $\Sigma_1^\sigma(n)$ and $\Sigma_1^{\tau_d}(n)$ are needed in order to obtain efficient membership algorithms.

3. New Characterizations of Boolean-Hierarchy Classes

In this section we develop new alternating-chain characterizations that allow the construction of efficient algorithms deciding membership for the Boolean hierarchies over Σ_1^σ, $\Sigma_1^{\tau_d}$, and Σ_2^ϱ for $|A| = 2$. We begin with the introduction of *marked words* and related partial orders which turn out to be crucial for the design of efficient algorithms.

3.1. Marked Words

For a fixed finite alphabet A, let $\mathcal{A} \stackrel{df}{=} \{\, [a,u] \mid a \in A, u \in A^* \,\}$ be the corresponding marked alphabet. Words over $\mathcal{A}$ are called marked words. For $w \in \mathcal{A}^*$ with $w = [a_1,u_1]\cdots[a_m,u_m]$ let $\overline{w} \stackrel{df}{=} a_1 \cdots a_m \in A^*$ be the corresponding unmarked word. Sometimes we use the functional notation $f_i(w) = a_1u_1^i \cdots a_mu_m^i$, i.e., $f_0(w) = \overline{w}$. Clearly, $f_0 : \mathcal{A}^* \to A^*$ is a surjection. For $x = x_1 \cdots x_m \in A^+$ and $u \in A^*$ we define $[x,u] \stackrel{df}{=} [x_1,\varepsilon]\cdots[x_{m-1},\varepsilon][x_m,u]$.

Next we define a relation on marked words. For $w, w' \in \mathcal{A}^*$ we write $w \preceq w'$ if and only if there exist $m \geq 0$, marked words $x_i, z_i \in \mathcal{A}^*$, and marked letters $b_i = [a_i,u_i] \in \mathcal{A}$ where $u_i \in A^+$ s.t.

$$\begin{array}{rcccccccccc} w & = & x_0b_1 & & x_1b_2 & & x_2 & \cdots & b_m & & x_m, \text{ and} \\ w' & = & x_0b_1 & z_1b_1 & x_1b_2 & z_2b_2 & x_2 & \cdots & b_m & z_mb_m & x_m. \end{array}$$

We call b_i the context letter of the insertion $z_i b_i$. We write $w \preceq^d w'$ if $w \preceq w'$ and $|f_0(z_i b_i)| \equiv 0 \pmod d$ for all i. Note that $\preceq^1$ coincides with $\preceq$ and observe that $\preceq^d$ is a transitive relation.

For a dfa $M = (A, Z, \delta, s_0, F)$ and $s, t \in Z$ we write $s \xrightarrow[M]{w} t$, if $\delta(s, \overline{w}) = t$ and for all i, $\delta(s, a_1 \cdots a_i) = \delta(s, a_1 \cdots a_i u_i)$. So $s \xrightarrow[M]{w} t$ means that the marked word w leads from s to t in a way such that the labels of w are consistent with loops in M. We say that w is M-consistent, if for some $t \in Z$, $s_0 \xrightarrow[M]{w} t$ and denote by $\mathcal{B}_M$ the set of marked words that are M-consistent. Every M-consistent word has the following nice property.

Proposition 3.1. *For $w = [c_1, u_1] \cdots [c_m, u_m] \in \mathcal{B}_M$ and all $j \geq 0$, $f_0(w) \equiv_M c_1 u_1^j \cdots c_m u_m^j$.*

3.2. New Characterization of the Classes $\boldsymbol{\Sigma_1^\sigma(n)}$ and $\boldsymbol{\Sigma_1^{\tau_{\mathrm{d}}}(n)}$

We extend the known characterization of the classes $\Sigma_1^{\tau_{\mathrm{d}}}(n)$ [Sel04] and add a characterization in terms of alternating chains on M-consistent marked words. Because we can also restrict the length of the labels u_i, we denote by $\mathcal{B}_M^c$ for any $c > 0$ the set of marked words $[a_0, u_0] \cdots [a_n, u_n]$ that are M-consistent and satisfy $|u_i| \leq c$ for all $i \leq n$.

Theorem 3.2. *The following is equivalent for $d, n \geq 1$, a dfa M, $c = |M|^{|M|}$, and $L = L(M) \subseteq A^+$.*

(1) $L \in \Sigma_1^{\tau_{\mathrm{d}}}(n)$
(2) $f_0^{-1}(L)$ *has no 1-alternating chain of length n in* $(\mathcal{B}_M; \preceq^d)$
(3) $f_0^{-1}(L)$ *has no 1-alternating chain of length n in* $(\mathcal{B}_M^c; \preceq^d)$

The case $d = 1$ is an alternative to the known characterization of the classes $\Sigma_1^\sigma(n)$ [GS01a].

Theorem 3.3. *Let M be a dfa, $L = L(M) \subseteq A^+$ and $n \geq 1$. Then $L \in \Sigma_1^\sigma(n)$ if and only if $f_0^{-1}(L)$ has no 1-alternating chain of length n in $(\mathcal{B}_M; \preceq)$.*

We can give an upper bound on d for languages in $\Sigma_1^\tau(n)$.

Theorem 3.4. *For every dfa M, $c = |M|^{|M|}$, and $d = c!$, $L(M) \in \Sigma_1^\tau(n) \Rightarrow L(M) \in \Sigma_1^{\tau_{\mathrm{d}}}(n)$.*

3.3. Characterization of the Classes $\boldsymbol{\Sigma_2^\varrho(n)}$ for $\boldsymbol{|A| = 2}$

We obtain an alternating-chain characterization for the classes of the Boolean hierarchy over Σ_2^ϱ for the case $|A| = 2$. This allows us to prove the first decidability result for this hierarchy. Note that only in case $|A| = 2$ decidability of $\mathrm{BC}(\Sigma_2^\varrho)$ [Str88] and Σ_3^ϱ [GS01b] is known.

For $u \in A^*$ let $\alpha(u)$ be the set of letters in u. We say that a marked word $w = [c_1, u_1] \cdots [c_m, u_m]$ satisfies the *alphabet condition* if for all $u_i \neq \epsilon$ it holds that $\alpha(u_i) = A$.

Theorem 3.5. *Let $A = \{a, b\}$, $n \geq 1$ and let $L(M) \subseteq A^*$ for some dfa M such that $L = L(M)$ is a star-free language. Then $L \in \Sigma_2^\varrho(n)$ if and only if $f_0^{-1}(L)$ has no 1-alternating chain $(w_0, \ldots, w_n)$ in $(\mathcal{B}_M; \preceq)$ such that all w_i satisfy the alphabet condition.*

4. Decidability and Complexity

The alternating-chain characterizations from the last sections can be used for the construction of efficient algorithms for testing the membership in these classes. As corollaries we obtain new decidability results: the classes $\Sigma_1^\tau(n)$ and $\Sigma_2^\varrho(n)$ for $|A| = 2$ are decidable.

The characterizations given in Theorems 3.2 and 3.3 allow the construction of nondeterministic, logarithmic-space membership tests for $\Sigma_1^\sigma(n)$ and $\Sigma_1^{\tau_{\mathrm{d}}}(n)$.

Theorem 4.1. *For all* $n \geq 1$, $\{M \,\big|\, M$ *is a det. finite automaton and* $L(M) \in \Sigma_1^\sigma(n)\} \in$ NL.

Theorem 4.2. *For all* $n \geq 1$, $\{M \,\big|\, M$ *is a det. finite automaton and* $L(M) \in \Sigma_1^{\tau_{\mathrm{d}}}(n)\} \in$ NL.

Remark 4.3. Unfortunately, we do not obtain NL-decidability for the classes $\Sigma_1^\tau(n)$. The reason is that the d in Theorem 3.4 is extremely big, i.e., we only know the upper bound $d \leq (m^m)!$ where m is the size of the automaton. We leave the question for an improved bound open. Note that if d can be bounded polynomially in the size of the automaton, then $\Sigma_1^\tau(n)$ is decidable in NL. Although d is very large, it is still computable from the automaton M which implies the decidability of all levels $\Sigma_1^\tau(n)$. This settles a question left open in [Sel04].

Theorem 4.4. *For all* $n \geq 1$, $\{M \,\big|\, M$ *is a det. finite automaton and* $L(M) \in \Sigma_1^\tau(n)\}$ *is decidable.*

Theorem 4.5. *For all* $n \geq 1$,
$\{M \,\big|\, M$ *is a det. finite automaton over the alphabet* $\{a, b\}$ *and* $L(M) \in \Sigma_2^\varrho(n)\} \in$ NL.

5. Exact Complexity Estimations

With the exception of $\Sigma_1^\tau(n)$, the membership problems of all classes of Boolean hierarchies considered in this paper are NL-complete. In contrast, the membership problems of the general classes FO_τ and FO_{τ_d} are PSPACE-complete and hence are strictly more complex.

Proposition 5.1. *Let* $\mathcal{C}$ *be any class of regular quasi-aperiodic languages over* A *with* $|A| \geq 2$ *and* $\emptyset \in \mathcal{C}$. *Then it is* NL*-hard to decide whether a given dfa* M *accepts a language in* $\mathcal{C}$.

Together with the upper bounds established in the previous sections this immediately implies the following exact complexity estimations.

Theorem 5.2. *Let* $k \geq 0$, $n \geq 1$, $d \geq 1$ *and* $\mathcal{C}$ *is one of the classes* $\mathcal{C}_k^d(n)$, $\Sigma_1^\sigma(n)$, $\Sigma_1^{\tau_{\mathrm{d}}}(n)$, *or* $\Sigma_2^\varrho(n)$ *for* $|A| = 2$. *Then* $\{M \,\big|\, M$ *is a det. finite automaton and* $L(M) \in \mathcal{C}\}$ *is* NL*-complete.*

We conclude this section with a corollary of the PSPACE-completeness of deciding FO_σ which was established by Stern [Ste85b] and by Cho and Huynh [CH91]. It shows that the complexity of deciding the classes FO_τ and FO_{τ_d} is strictly higher than the complexity of deciding the classes mentioned in Theorem 5.2. (Note that NL is closed under logspace many-one reductions, NL $\subseteq$ DSPACE($\log^2 n$) [Sav70] and DSPACE($\log^2 n$) $\subsetneq$ PSPACE [HS65]. Hence the classes FO_τ and FO_{τ_d} can not be decided in NL.)

Theorem 5.3. *The classes* FO_τ *and* FO_{τ_d} *are* PSPACE*-complete.*

6. Conclusions

The results of this paper (as well as several previous facts that appeared in the literature) show that more and more decidable levels of hierarchies turn out to be decidable in NL. One is tempted to strengthen the well-known challenging conjecture of decidability of the dot-depth hierarchy to the conjecture that all levels of reasonable hierarchies of first-order definable regular languages are decidable in NL. At least, it seems instructive to ask this question about any level of such a hierarchy known to be decidable.

In this paper we considered the complexity of classes of regular languages only w.r.t. the representation of regular languages by dfa's. Similar questions are probably open for other natural representations of regular languages, like nondeterministic finite automata and propositions of monadic second order, first order or temporal logics.

Acknowledgements

We are grateful to Klaus W. Wagner for many helpful discussions. We would like to thank the anonymous referees for their valuable comments.

References

[BCST92] D. A. Mix Barrington, K. Compton, H. Straubing, and D. Thérien. Regular languages in NC^1. *J. Computer and System Sciences*, 44:478–499, 1992.

[Brz76] J. A. Brzozowski. Hierarchies of aperiodic languages. *RAIRO Inform. Theor.*, 10:33–49, 1976.

[Büc60] J. R. Büchi. Weak second-order arithmetic and finite automata. In *Z. Math. Logik und grundl. Math.*, 6:66-92, 1960.

[CH91] S. Cho and D. T. Huynh. Finite-automaton aperiodicity is PSPACE-complete. *Theoret. Computer Science*, 88:99–116, 1991.

[CB71] R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *J. Computer and System Sciences*, 5:1–16, 1971.

[CPP93] J. Cohen, D. Perrin, and J. E. Pin. On the expressive power of temporal logic. *J. Computer and System Sciences*, 46:271–294, 1993.

[CPS06] L. Chaubard, J. E. Pin, and H. Straubing. First order formulas with modular predicates. In *Proc. 21th IEEE Symposium on Logic in Computer Science*, pp. 211–220. IEEE Comp. Society, 2006.

[GS00a] C. Glaßer and H. Schmitz. Languages of dot-depth 3/2. In *Proc. 17th STACS*, Lect. Notes in Comp. Science **1770**, pp. 555–566. Springer, 2000.

[GS00b] C. Glaßer and H. Schmitz. Decidable hierarchies of starfree languages. In *Proc. 20th FST-TCS*, Lect. Notes in Comp. Science **1974**, pp. 503–515. Springer, 2000.

[GS01a] C. Glaßer and H. Schmitz. The boolean structure of dot-depth one. *Journal of Automata, Languages and Combinatorics*, 6(4):437–452, 2001.

[GS01b] C. Glaßer and H. Schmitz. Level 5/2 of the Straubing-Thérien hierarchy for two-letter alphabets. In *Preproceedings 5th Conference on Developments in Language Theory*, pp. 254–265, 2001.

[GSS07] C. Glaßer, H. Schmitz and V. Selivanov. Efficient algorithms for membership in Boolean hierarchies of regular languages. ECCC Report TR07-094, October 2007.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Computing*, 17:935–938, 1988.

[Kna83] R. Knast. A semigroup characterization of dot-depth one languages. *RAIRO Inform. Theor.*, 17:321–330, 1983.

[MPT00] A. Maciel, P. Péladeau, and D. Thérien. Programs over semigroups of dot–depth one. *Theoret. Computer Science*, 245:135–148, 2000.

[Pin95] J. E. Pin. Finite semigroups and recognizable languages: an introduction. In J. Fountain, editor, *NATO Advanced Study Institute: Semigroups, Formal Languages and Groups*, pp. 1–32. Kluwer Academic Publishers, 1995.

[Pin96a] J. E. Pin. Logic, semigroups and automata on words. *Annals of Mathematics and Artificial Intelligence*, 16:343–384, 1996.

[Pin96b] J. E. Pin. Syntactic semigroups. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume I, pp. 679–746. Springer, 1996.

[PW02] J. E. Pin and P. Weil. The wreath product principle for ordered semigroups. *Communications in Algebra*, 30:5677–5713, 2002.

[PP86] D. Perrin and J. E. Pin. First-order logic and star-free sets. *Journal of Computer and System Sciences*, 32:393–406, 1986.

[Sav70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sch01] H. Schmitz. *The Forbidden-Pattern Approach to Concatenation Hierarchies*. PhD thesis, Fakultät für Mathematik und Informatik, Universität Würzburg, 2001.

[Sel01] V. L. Selivanov. A logical approach to decidability of hierarchies of regular star-free languages. In *Proc. 18th STACS*, Lect. Notes in Comp. Science **2010**, pp. 539–550. Springer, 2001.

[Sel04] V. L. Selivanov. Some hierarchies and reducibilities on regular languages. Technical Report 349, Inst. für Informatik, Univ. Würzburg, 2004.

[Shu98] A.G. Shukin. Difference hierarchies of regular languages (in russian). *Computing Systems (Novosibirsk, Institute of Mathematics)*, (161):141–155, 1998.

[Sim75] I. Simon. Piecewise testable events. In *Proceedings 2nd GI Conference*, Lect. Notes in Comp. Science **33**, pp. 214–222. Springer, 1975.

[SS00] V. L. Selivanov and A. G. Shukin. On hierarchies of regular star-free languages. Technical Report Preprint 69, A. P. Ershov Institute of Informatics Systems, Novosibirsk, 2000.

[Ste85a] J. Stern. Characterizations of some classes of regular events. *Theoret. Computer Science*, 35:17–42, 1985.

[Ste85b] J. Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66:163–176, 1985.

[Str81] H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theoret. Computer Science*, 13:137–150, 1981.

[Str85] H. Straubing. Finite semigroup varieties of the form $\mathbf{V} * \mathbf{D}$. *J.Pure Appl.Algebra*, 36:53–94, 1985.

[Str88] H. Straubing. Semigroups and languages of dot-depth two. *Theoret. Computer Science*, 58:361–378, 1988.

[Str94] H. Straubing. *Finite automata, formal logic and circuit complexity*. Birkhäuser, Boston, 1994.

[SW98] H. Schmitz and K. W. Wagner. The Boolean hierarchy over level 1/2 of the Straubing-Thérien hierarchy. `http://arxiv.org/abs/cs.CC/9809118`.

[Sze87] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Bull. of the EATCS*, 33:96–100, 1987.

[Thé81] D. Thérien. Classification of finite monoids: the language approach. *Theoret. Computer Science*, 14:195–208, 1981.

[Tho82] W. Thomas. Classifying regular events in symbolic logic. *J. Comp. and System Sciences*, 25:360–376, 1982.

[Tra58] B. A. Trakhtenbrot. Synthesis of logic networks whose operators are described by means of single-place predicate calculus. *Doklady Akad. Nauk SSSR*, 118:646–649, 1958.

[Wei04] P. Weil. Algebraic recognizability of languages. In *Proc. 29th Mathematical Foundations of Computer Science*, *Lect. Notes in Comp. Science* **3153**, pp. 149–175. Springer, 2004.

[Yu96] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume I, pp. 41–110. Springer, 1996.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 349-360
www.stacs-conf.org

ON THE COMPLEXITY OF ELEMENTARY MODAL LOGICS

EDITH HEMASPAANDRA AND HENNING SCHNOOR

Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623, U.S.A.
E-mail address, Edith Hemaspaandra: eh@cs.rit.edu
E-mail address, Henning Schnoor: hs@cs.rit.edu

Abstract. Modal logics are widely used in computer science. The complexity of modal satisfiability problems has been investigated since the 1970s, usually proving results on a case-by-case basis. We prove a very general classification for a wide class of relevant logics: Many important subclasses of modal logics can be obtained by restricting the allowed models with first-order Horn formulas. We show that the satisfiability problem for each of these logics is either NP-complete or PSPACE-hard, and exhibit a simple classification criterion. Further, we prove matching PSPACE upper bounds for many of the PSPACE-hard logics.

1. Introduction

Modal logics have proven to be a valuable tool in mathematics and computer science. The traditional uni-modal logic enriches the propositional language with the operator $\Diamond$, where $\Diamond\varphi$ is interpreted as φ *possibly holds*. The usual semantics interpret modal formulas over graphs, where $\Diamond\varphi$ means "there is a successor world where φ is true." In addition to their mathematical interest, modal logics are widely used in practical applications: In artificial intelligence, modal logic is used to model the knowledge and beliefs of an agent, see e.g. [BZ05]. Modal logics also can be applied in cryptographic and other protocols [FHJ02, CDF03, HMT88, LR86]. For many specific applications, there exist tailor-made variants of modal logics [BG04].

Due to the vast number of applications, complexity issues for modal logics are very relevant, and have been examined since Ladner's seminal work [Lad77]. Depending on the application, modal logics with different properties are studied. For example, one might want the formula $\varphi \to \Diamond\varphi$ to be an axiom—if something is true, then it should be considered possible. Or $\Diamond\Diamond\varphi \to \Diamond\varphi$—if it is possible that φ is possible, then φ itself should be possible. Classical results [Sah73] show that there is a close correspondence between modal logics defined by axioms and logics obtained by restricting the class of considered graphs. Requiring the axioms mentioned above corresponds to restricting the classes of graphs to those which are reflexive or transitive, respectively. Determining the complexity of a given

Supported in part by NSF grants CCR-0311021 and IIS-0713061, a Friedrich Wilhelm Bessel Research Award, and the DAAD postdoc program.

modal logic, defined either by the class of considered graphs or via a modal axiom system, has been an active line of research since Ladner's results. In particular, the complexity classes NP and PSPACE have been at the center of attention.

Most complexity results have been on a case-by-case basis, proving results for individual logics both for standard modal logics and variations like temporal or hybrid logics [HM92, Ngu05, SC85]. Examples of more general results include Halpern and Rêgo's proof that logics including the *negative introspection* axiom, which corresponds to the Euclidean graph property, have an NP-complete satisfiability problem [HR07]. In [SP06], Schröder and Pattinson show a way to prove PSPACE upper bounds for modal logics defined by modal axioms of modal depth 1. In [Lad77], Ladner proved PSPACE-hardness for all logics for which reflexive and transitive graphs are admissible models. In [Spa93], Hemaspaandra showed that all normal logics extending S4.3 have an NP-complete satisfiability problem, and work on the *Guarded Fragment* has shown that some classes of modal logics can be seen as a decidable fragment of first-order logic [AvBN98].

While these results give hardness or upper bounds for classes of logics, they do not provide a full case distinction identifying *all* "easy" or "hard" cases in the considered class. We achieve such a result: For a large class of modal logics containing many important representatives, we identify *all* cases which have an NP-complete satisfiability problem, and show that the satisfiability problem for *all* other non-trivial logics in that class is PSPACE-hard. Hence these problems avoid the infinitely many complexity classes between NP and PSPACE, many of which have natural complete problems arising from logical questions. To our knowledge, such a general result has not been achieved before.

To describe the considered class of modal logics, note that many relevant properties of modal models can be expressed by first-order formulas: A graph is transitive if its edge-relation R satisfies the clause $\forall xyz\,(xRy \wedge yRz \rightarrow xRz)$ and symmetric if it satisfies $\forall xy\,(xRy \rightarrow yRx)$. Many other graph properties can be defined using similar formulas, where the presence of a certain pattern of edges in the graph forces the existence of another. Analogously to propositional logic, we call conjunctions of such clauses *universal Horn formulas*. Many relevant logics can be defined in this way: All examples form [Lad77] fall into this category, as well as logics over Euclidean graphs.

We study the following problem: Given a universal Horn formula $\hat{\psi}$, what is the complexity of the modal satisfiability problem over the class of graphs defined by $\hat{\psi}$?

The main results of this paper are the following: First, we identify all cases which give a satisfiability problem solvable in NP (which then for every nontrivial logic is NP-complete), and show that all other cases are PSPACE-hard. Second, we prove a generalization of a "tree-like model property," and use it to obtain PSPACE upper bounds for a large class of logics. As a corollary, we prove that Ladner's classic hardness result is "optimal" in the class of logics defined by universal Horn formulas. A further corollary is that in the universal Horn class, all logics whose satisfiability problem is not PSPACE-hard already have the "polynomial-size model property," which is only one of several known ways to prove NP upper bounds for modal logics.

Various work was done on restricting the syntax of the modal formulas by restricting the propositional operators [BHSS06], the nesting degree and number of variables [Hal95] or considering modal formulas in Horn form [CL94]. While these results are about restricting the *syntax* of the modal formulas, the current work studies different *semantics* of modal logics, where the semantics are specified by Horn formulas.

logic name	graph property	formula definition
K	All graphs	$\mathsf{K}(\hat{\varphi}_{\text{taut}})$
T	reflexive	$\mathsf{K}(\hat{\varphi}_{\text{refl}})$
B	symmetric	$\mathsf{K}(\hat{\varphi}_{\text{symm}})$
K4	transitive graphs	$\mathsf{K}(\hat{\varphi}_{\text{trans}})$
S4	transitive and reflexive	$\mathsf{K}(\hat{\varphi}_{\text{trans}} \wedge \hat{\varphi}_{\text{refl}})$
S5	equivalence relations	$\mathsf{K}(\hat{\varphi}_{\text{trans}} \wedge \hat{\varphi}_{\text{refl}} \wedge \hat{\varphi}_{\text{symm}})$

Table 1: Common modal logics

In Section 2, we introduce terminology and generalize classic complexity results. Section 3 introduces universal Horn formulas and states our main result. In Section 4, we explain the main ideas of the proof, before we obtain matching PSPACE upper bounds for many of the involved logics in Section 5, generalizing many previously known results. Proofs and precise definitions can be found in the full version of the paper.

2. Preliminaries

2.1. Basic Concepts and Notation

Modal logic is an extension of propositional logic. A modal formula is a propositional formula using variables, the usual logical symbols $\wedge, \vee, \neg$, and a unary operator $\Diamond$. (A dual operator $\Box$ is often considered as well, this can be regarded as abbreviation for $\neg\Diamond\neg$.) A model for a modal formula is a set of connected "worlds" with individual propositional assignments. To be precise, a *frame* is a directed graph $G = (W, R)$, where the vertices in W are called "worlds," and an edge $(u, v) \in R$ is interpreted as v is "considered possible" from u. A *model* $M = (G, X, \pi)$ consists of a frame $G = (W, R)$, a set X of propositional variables and a function π assigning to each variable $x \in X$ a subset of W, the set of worlds in which x is true. We say the model M is *based* on the frame (W, R). If $\mathcal{F}$ is a class of frames, then a model is an $\mathcal{F}$-model if it is based on a frame in $\mathcal{F}$. With $|M|$ we denote the number of worlds in M. For a world $w \in W$, we define when a modal formula ϕ is *satisfied* at w in M (written $M, w \models \phi$). If ϕ is a variable x, then $M, w \models \phi$ if and only if $w \in \pi(x)$. As usual, $M, w \models \phi_1 \wedge \phi_2$ if and only if $M, w \models \phi_1$ and $M, w \models \phi_2$, and $M, w \models \neg\phi$ iff $M, w \not\models \phi$. For the modal operator, $M, w \models \Diamond\phi$ if and only if there is a world $w' \in W$ such that $(w, w') \in R$ and $M, w' \models \phi$.

We describe a way to define classes of frames with propositional formulas. The *frame language* is the first-order language containing (in addition to the operators $\wedge, \vee$, and $\neg$) the binary relation R, which is interpreted as the edge relation in a graph. Semantics are defined in the obvious way, e.g., a graph satisfies the formula $\hat{\varphi}_{\text{trans}} := \forall xyz(xRy \wedge yRz \rightarrow xRz)$ if and only if it is transitive. In order to separate modal formulas from first-order formulas, we use $\hat{\cdot}$ to denote the latter, e.g., $\hat{\varphi}$ is a first-order formula, while ϕ is a modal formula.

A modal logic usually is defined as the set of the formulas provable in it. Since a formula is satisfiable iff its negation is not provable, we can define a logic by the set of formulas satisfiable in it. For a first-order formula $\hat{\psi}$ over the frame language, we define the logic $\mathsf{K}(\hat{\psi})$ as the logic in which a modal formula ϕ is satisfiable if and only if there is a model M and a world $w \in M$ such that the frame which M is based on satisfies $\hat{\psi}$ (we simply

write $M \models \hat{\psi}$ for this), and $M, w \models \phi$. Many relevant modal logics can be expressed in this way: In addition to the formula $\hat{\varphi}_{\text{trans}}$ defined above, let $\hat{\varphi}_{\text{refl}} := \forall w(wRw)$, and let $\hat{\varphi}_{\text{symm}} := \forall xy(xRy \to yRx)$. Finally, let $\hat{\varphi}_{\text{taut}}$ be some tautology over the frame language. Table 1 introduces common modal logics and how they can be expressed in our framework. For a formula $\hat{\psi}$ over the frame language, we consider the following problem:

Problem: $\mathsf{K}(\hat{\psi})$-SAT
Input: A modal formula ϕ
Question: Is ϕ satisfiable in a model based on a frame satisfying $\hat{\psi}$?

As an example, the problem $\mathsf{K}(\hat{\varphi}_{\text{trans}})$-SAT is the problem of deciding if a given modal formula can be satisfied in a transitive frame, and is the same as the satisfiability problem for the logic K4. In the problem $\mathsf{K}(\hat{\psi})$-SAT, the formula $\hat{\psi}$ is fixed. It is also interesting to study the *uniform* version of the problem, where we are given a formula $\hat{\psi}$ over the frame language and a modal formula ϕ, and the goal is to determine whether there exists a model satisfying both. This problem obviously is PSPACE-hard (this easily follows from Theorem 2.2). In this paper, we study the complexity behavior of *fixed* modal logics, and focus on the NP-PSPACE-gap in complexity. The property of having "small models" is often crucial, as these often lead to a satisfiability problem in NP. A modal logic KL has the *polynomial-size model property*, if there is a polynomial p, such that for every KL-satisfiable formula ϕ, there is a KL-model M and a world $w \in M$ such that $M, w \models \phi$, and $|M| \leq p(|\phi|)$. Since modal logic is an extension of propositional logic, the satisfiability problem for every non-trivial modal logic is NP-hard. Hence, showing that a modal logic has a satisfiability problem in NP is an optimal complexity bound. The following standard observation is the basis of our NP containment proofs:

Proposition 2.1. *Let $\hat{\psi}$ be a first-order formula over the frame language, such that $\mathsf{K}(\hat{\psi})$ has the polynomial-size model property. Then $\mathsf{K}(\hat{\psi})$-SAT $\in$ NP.*

2.2. Ladner's Theorem and Applications

In [Lad77], Ladner proved complexity results for a variety of modal logics. An *extension* of a modal logic KL is a modal logic KL′ such that every formula which is valid in KL is also valid in KL′, or equivalently such that every KL′-satisfiable formula is KL-satisfiable. For example, every logic that we consider is an extension of K, and S4 is an extension of K4. It is easy to see that if $\hat{\psi}_1$ and $\hat{\psi}_2$ are formulas over the frame language such that $\hat{\psi}_1$ implies $\hat{\psi}_2$, then $\mathsf{K}(\hat{\psi}_1)$ is an extension of $\mathsf{K}(\hat{\psi}_2)$. Ladner's main result can be stated as follows.

Theorem 2.2 ([Lad77])**.** *(1) The satisfiability problems for* K, K4, *and* S4 *are* PSPACE*-complete, and* S5-SAT *is* NP*-complete.*
(2) If S4 *is an extension of* KL*, and* KL *extends* K*, then* KL-SAT *is* PSPACE*-hard.*

The ideas from Ladner's proof for Theorem 2.2 can be extended to show similar results. We define a generalization of transitivity. For a number k, we say that a graph G is k-transitive if for every pair of vertices u, v in G such that there is a k-step path from u to v in G, there is an edge (u, v). Note that a graph is transitive if and only if it is 2-transitive. For a set $S \subseteq \mathbb{N}$, a graph is S-transitive if it is k-transitive for every $k \in S$. A *strict tree* is a directed connected graph which has a root w from which all other vertices can

be reached via a unique path. A reflexive/S-transitive/symmetric tree is the reflexive/S-transitive/symmetric closure of a strict tree.

Theorem 2.3. *Let $\hat{\psi}$ be a first-order formula over the frame language such that one of the following cases applies:*

- *$\hat{\psi}$ is satisfied in every strict tree,*
- *$\hat{\psi}$ is satisfied in every reflexive tree,*
- *there is a set $S \subseteq \mathbb{N}$ such that $\hat{\psi}$ is satisfied in every S-transitive tree,*
- *$\hat{\psi}$ is satisfied in every symmetric tree,*
- *$\hat{\psi}$ is satisfied in every tree which is both reflexive and symmetric,*
- *there is a set $S \subseteq \mathbb{N}$ such that $\hat{\psi}$ is satisfied in every tree which is both reflexive and S-transitive.*

Then $\mathsf{K}(\hat{\psi})$-SAT is PSPACE*-hard and $\mathsf{K}(\hat{\psi})$ does not have the polynomial-size model property.*

This theorem follows with straightforward observations from Ladner's proof, by applying the well-known tree-model property for the involved logics, and using similar ideas for the case of symmetric models. In the next section, we will present the main result of this paper: In the class of modal logics defined by universal Horn formulas, all non-trivial cases that are not covered by Theorem 2.3 have an NP-complete satisfiability problem.

3. Universal Horn Formulas and the Main Result

We now consider a syntactically restricted case of universal first-order formulas, namely Horn formulas. Many well-known modal logics can be expressed in this way. Usually, a Horn clause is defined as a disjunction of literals of which at most one is positive. If a positive literal occurs, then the clause $\overline{x_1} \vee \cdots \vee \overline{x_n} \vee y$ can be written as the implication $x_1 \wedge \cdots \wedge x_n \rightarrow y$. Since in the context of the frame language, an atomic proposition is of the form xRy, the following is the natural version of Horn clauses for our purposes:

Definition 3.1. A *universal Horn clause* over the basic frame language is a formula of the form $x_1^1 R x_2^1 \wedge \cdots \wedge x_1^k R x_2^k \rightarrow C$, where C is of the form xRy or $C = \mathsf{false}$, where all (not necessarily distinct) variables are implicitly universally quantified.

A *universal Horn formula* is a conjunction of universal Horn clauses. Due to space reasons, in this paper we only consider Horn clauses of the first form (known as positive Horn clauses), our results hold for the second form analogously. With universal Horn formulas, many of the frame properties usually considered can be expressed, like transitivity, symmetry, euclidicity, etc. We now state the classification theorem:

Theorem 3.2. *Let $\hat{\psi}$ be a universal Horn formula. Then either $\hat{\psi}$ satisfies the condition from Theorem 2.3, (and thus $\mathsf{K}(\hat{\psi})$-SAT is* PSPACE*-hard) or $\mathsf{K}(\hat{\psi})$ has the polynomial-size model property and $\mathsf{K}(\hat{\psi})$-$\mathsf{SAT} \in$* NP*.*

4. A Proof Sketch

We now give an overview of the ideas used to prove our main result. We first show that universal Horn clauses can be represented as graphs in such a way that the properties of the involved logics can be characterized with homomorphic images of the defining Horn

clauses. In Section 4.2, we then demonstrate at an example how NP results can be shown for universal Horn clauses, before outlining the strategy for the proof of Theorem 3.2.

4.1. Universal Horn clauses and homomorphisms

For a universal Horn clause $\hat{\varphi} = x_1^1 R x_2^1 \wedge \cdots \wedge x_1^k R x_2^k \to xRy$, the *prerequisite graph of* $\hat{\varphi}$, denoted with $\mathrm{prereq}(\hat{\varphi})$, consists of the variables appearing on the left-hand side of the implication $\hat{\varphi}$ as vertices, where for variables $u, v \in \{x_1^1, \ldots, x_2^k\}$, there is an edge (u, v) if the clause uRy appears on the left-hand side of the formula. The *conclusion edge of* $\hat{\varphi}$, denoted with $\mathrm{conc}(\hat{\varphi})$, is the edge (x, y). As usual, a *homomorphism* between graphs is a function on the vertices preserving the edge relation. The definition of the prerequisite graph and the conclusion edge of a universal Horn formula establishes a one-to-one correspondence between universal Horn clauses and their representation as graphs. These definitions allow us to relate truth of a Horn clause to homomorphic images of the involved graphs. In the following, $\mathrm{prereq}(\hat{\varphi}) \cup \{x, y\}$ is the graph obtained from $\mathrm{prereq}(\hat{\varphi})$ by adding (if not already present) the vertices x and y, but no additional edges.

Proposition 4.1. *Let $\hat{\varphi}$ be a universal Horn clause with $\mathrm{conc}(\hat{\varphi}) = (x, y)$. A graph G satisfies $\hat{\varphi}$ if and only if for every homomorphism $\alpha\colon \mathrm{prereq}(\hat{\varphi}) \cup \{x, y\} \to G$, there is an edge $(\alpha(x), \alpha(y))$ in G.*

This observation is central, as it shows that properties of a logic $\mathsf{K}(\hat{\varphi})$ depend on the types of homomorphic images of $\mathrm{prereq}(\hat{\varphi})$.

4.2. Example of a Case in NP

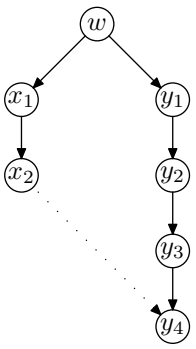


Figure 1: $\hat{\varphi}^{2\to 4}$

We now give an example of the proof of NP membership. Let $\hat{\varphi}^{k\to l}$ be the formula $wRx_1 \wedge x_1Rx_2 \wedge \cdots \wedge x_{k-1}Rx_k \wedge wRy_1 \wedge y_1Ry_2 \wedge \cdots \wedge y_{l-1}Ry_l \to x_kRy_l$, where all variables are universally quantified (and in the case that $k = 0$ or $l = 0$, we replace x_0 or y_0 with w). In Figure 1, we present the graph representation of the formula $\hat{\varphi}^{2\to 4}$. A graph G satisfies $\hat{\varphi}^{k\to l}$ if and only if for any nodes $w, x_k, y_l \in G$, if there is a k-step path from w to x_k and an l-step path from w to y_l, then (x_k, y_l) is an edge in G. This definition generalizes several well-known examples: A graph is reflexive if and only if it satisfies $\hat{\varphi}^{0\to 0}$, symmetry is expressed with $\hat{\varphi}^{1\to 0}$, and k-transitivity with $\hat{\varphi}^{0\to k}$. A graph is Euclidean iff it satisfies $\hat{\varphi}^{1\to 1}$. Thus, this notation allows us to capture many interesting graph properties. We give the proof idea for a relatively easy special case. Generalizations of this idea are the main ingredients for our polynomial-size model proofs.

Theorem 4.2. *Let $k \geq 1$, and let $\hat{\psi}$ be a universal formula over the frame language such that $\hat{\psi}$ implies $\hat{\varphi}^{k\to k}$. Then $\mathsf{K}(\hat{\psi})$ has the polynomial-size model property, and $\mathsf{K}(\hat{\psi})$-SAT $\in$ NP.*

Proof Sketch. It suffices to prove the polynomial-size model property. We need to show that every $\mathsf{K}(\hat{\psi})$-satisfiable formula ϕ has a "small" model. Let M be a model and w a world such that $M, w \models \phi$, and M also satisfies the first-order formula $\hat{\psi}$, in particular it then satisfies $\hat{\varphi}^{k\to k}$, which means that for t, u, v in M, if there is a k-step path from t to u and also from t to v, then (u, v) is an edge in M. Note that since $\hat{\psi}$ is universal, every restriction of M still satisfies $\hat{\psi}$. With a standard construction, we can restrict the number of vertices in M which do not have a k-step predecessor to polynomial size.

We show that the edge relation restricted to the set C of nodes having k-step predecessors is an equivalence relation, then with a standard argument (similar to the proof for the logic $\mathsf{S5}$), we can pick one world for every satisfied subformula of ϕ from polynomially many "clusters" in C, and obtain a small submodel M' of M satisfying $M', w \models \phi$.

We first show that every node v in C is reflexive: It has a k-step predecessor t, and hence by the $\hat{\varphi}^{k\to k}$-property, there is an edge (v, v). We show that C is symmetric: Let (u, v) be an edge in C. From the above, we know that both of these nodes are reflexive in C. Hence there is a k-step path from u to both u and v. From the $\hat{\varphi}^{k\to k}$-property, it follows that (v, u) is an edge. Finally, we prove that C is transitive. Let (t, u) and (u, v) be edges in C. Since we already showed that C is symmetric, we know that (u, t) is an edge as well. Since all of the involved nodes are also reflexive, there is a k-step path from u to both t and v, implying that (t, v) is an edge as well. □

The analog of Theorem 4.2 can be shown to hold for many other cases, for example showing that all logics of the form $\mathsf{K}(\hat{\varphi}^{k\to l})$ for $1 \leq k, l$ or $k \geq 2, l = 0$ have satisfiability problems in NP. Note that the case $k = 1$ of Theorem 4.2 follows from the main result of [HR07]. Our results and theirs are incomparable: They achieve the NP result for *all* modal logics extending what in our notation is $\mathsf{K}(\hat{\varphi}^{1\to 1})$, where our results only hold for logics defined by universal formulas. On the other hand, Theorem 3.2 gives NP membership for a large class of logics which do not follow from their result, namely all logics defined by universal Horn formulas not satisfied on the various forms of trees mentioned in Theorem 2.3.

Similarly to Proposition 4.1, it can be shown that there is a close relationship between implications of Horn clauses and homomorphisms between their prerequisite graphs. This relationship and Theorem 4.2 imply that every universal Horn clause which can be homomorphically mapped into the clause $\hat{\varphi}^{k\to k}$ in an appropriate way defines a logic having the polynomial-size model property and a satisfiability problem in NP. Therefore we can collect "homomorphism properties" of Horn clauses that guarantee this property of the corresponding logics (we need to be careful with what a homomorphism exactly is in this context—we will not go into the technical details here). We can define a class $\mathcal{H}_{\mathrm{NP}}$ of graphs, containing among others representations of generalizations of $\hat{\varphi}^{k\to l}$ for those k, l leading to the polynomial-size model property, which has the following properties:

(1) For all universal Horn clauses $\hat{\varphi}$ which can be homomorphically mapped into an element of $\mathcal{H}_{\mathrm{NP}}$, $\mathsf{K}(\hat{\varphi})$ has the polynomial-size model property, and $\mathsf{K}(\hat{\varphi})$-$\mathsf{SAT} \in \mathrm{NP}$.
(2) $\mathcal{H}_{\mathrm{NP}}$ is large enough to form the basic building blocks of almost all of our NP results.

An example for a graph in $\mathcal{H}_{\mathrm{NP}}$ is the following: If a clause $\hat{\varphi}$ with $\mathrm{conc}(\hat{\varphi}) = (x, y)$ can be mapped into a "generalized line" $(l_0, \ldots, l_n)$ with a homomorphism α satisfying $\alpha(y) = l_i$ and $\alpha(x) = l_{i+k}$ for $k \geq 2$, then the NP conditions are met. This is satisfied, for example, for clauses of the form $\hat{\varphi}^{k\to 0}$ for $k \geq 2$. While the set $\mathcal{H}_{\mathrm{NP}}$ contains a large class of graphs, on its own it is not sufficient to prove all NP results for logics defined by universal Horn formulas. The main reason is that it does not take into account interference between clauses in Horn formulas. For example, it is well known that reflexivity, symmetry, and transitivity alone lead to PSPACE-complete logics. But the combination of these requirements defines the logic $\mathsf{S5}$, which has a satisfiability problem in NP. Hence, interference between clauses in a Horn formulas is of crucial importance for the complexity of the defined logic.

4.3. An Alternative Formulation and Proof Idea of the Main Theorem

We give an idea of the main strategy used to prove the classification Theorem 3.2. For this, we restate the complexity classification as the algorithm Horn-Classification (see Figure 3) which, given a universal Horn formula $\hat{\psi}$ as input, determines the complexity of $\mathsf{K}(\hat{\psi})$-SAT. The output of the algorithm and the statement of Theorem 3.2 agree in all cases. The purpose of the algorithm is not to be implemented, but to serve as a case distinction used to prove Theorem 3.2, and several later corollaries.

The idea of the algorithm is the following: In *types-list*, it maintains a list of implications of the formula $\hat{\psi}$. For example, Horn-Classification puts `refl` into *types-list* if it detects the formula $\hat{\psi}$ to require any graph G satisfying it to be "near reflexive" (every vertex having long enough paths ending in and originating in it must be reflexive). Similarly, $\texttt{symm} \in$ *types-list* ($\texttt{trans}^k \in$ *types-list*) means that $\hat{\psi}$ requires a graph to be "near symmetric" ("near k-transitive"). The occurring class $\mathcal{H}_{\mathrm{NP}}^{\textit{types-list}}$ is a generalization of the set $\mathcal{H}_{\mathrm{NP}}$ introduced earlier. Since we are not considering only individual clauses anymore, this class is not constant, but dynamically grows corresponding to collected implications of the formula kept in *types-list*. For example, $\mathcal{H}_{\mathrm{NP}}^{\{\texttt{refl}\}}$ contains reflexive closures of elements in $\mathcal{H}_{\mathrm{NP}}$.

If Horn-Classification detects that a clause $\hat{\varphi}$ can be mapped into an element of $\mathcal{H}_{\mathrm{NP}}^{\textit{types-list}}$, then the clause $\hat{\varphi}$, in addition with the requirements kept in *types-list*, implies the polynomial-size model property. Another condition leading to NP containment of the logic is the following: It is well known that the modal logic over the class of frames which are both transitive and symmetric has a satisfiability problem in NP. Generalizing this, when Horn-Classification detects that $\hat{\psi}$ implies "near symmetry" and "near k-transitivity," this also leads to NP solvability of the satisfiability problem.

For a set *types-list* $\subseteq \{\texttt{refl}, \texttt{symm}, \texttt{trans}^k \mid k \in \mathbb{N}\}$, a graph G satisfies the conditions of *types-list* if it has the corresponding properties, i.e., if $\texttt{refl} \in$ *types-list* ($\texttt{symm} \in$ *types-list*, $\texttt{trans}^k \in$ *types-list*), then G is required to be reflexive (symmetric, k-transitive). A *types-list* tree is obtained from a strict tree by adding exactly those edges required to make it satisfy the conditions of *types-list* (this is a natural closure operator). For a universal Horn clause $\hat{\varphi}$, let *types-list*-$T_{\hat{\varphi}}^{\mathrm{hom}}$ denote the pairs (α, T) such that T is a *types-list* tree, and $\alpha\colon \mathrm{prereq}(\hat{\varphi}) \to T$ is a homomorphism. Due to Proposition 4.1, this is the set of *types-list* trees about which the clause $\hat{\varphi}$ "makes a statement," along with the corresponding homomorphisms.

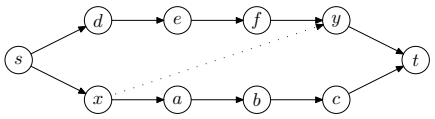


Figure 2: Example Formula

We now define the properties of Horn clauses which do not lead to NP containment of the satisfiability problems on their own. Recalling Section 2.2, it is natural that clauses which are satisfied in every reflexive, transitive, or symmetric tree are among these. This is captured by the following definitions: If $\hat{\varphi}$ is a universal Horn clause with $\mathrm{conc}(\hat{\varphi}) = (x, y)$, we say that ($\hat{\varphi}$, *types-list*) satisfies the *reflexive case*, if for every $(\alpha, T) \in$ *types-list*-$T_{\hat{\varphi}}^{\mathrm{hom}}$ it holds that $\alpha(x) = \alpha(y)$. ($\hat{\varphi}$, *types-list*) satisfies the *transitive case for* $k \in \mathbb{N}$, if for every $(\alpha, T) \in$ *types-list*-$T_{\hat{\varphi}}^{\mathrm{hom}}$ there is a path from $\alpha(x)$ to $\alpha(y)$ in T, and there is some $(\alpha, T) \in$ *types-list*-$T_{\hat{\varphi}}^{\mathrm{hom}}$ such that $\alpha(y)$ is exactly k levels below $\alpha(x)$ in T. Finally, ($\hat{\varphi}$, *types-list*) satisfies the *symmetric case* if for every $(\alpha, T) \in$ *types-list*-$T_{\hat{\varphi}}^{\mathrm{hom}}$, there

Input: Universal Horn formula $\hat{\psi}$
types-list $:= \emptyset$
while not done **do**
if every clause in $\hat{\psi}$ is satisfied on every *types-list* tree **then**
$\mathsf{K}(\hat{\psi})$-SAT is PSPACE-hard
end if
Let $\hat{\varphi}$ be a clause in $\hat{\psi}$ not satisfied on every *types-list* tree
if $\hat{\varphi}$ can homomorphically be mapped into a graph from $\mathcal{H}_{\mathrm{NP}}^{\textit{types-list}}$ **then**
$\mathsf{K}(\hat{\psi})$ has the polynomial-size model property, and $\mathsf{K}(\hat{\psi})$-SAT $\in$ NP.
else
if $(\hat{\varphi}, \textit{types-list})$ satisfies the reflexive case **then**
types-list $:=$ *types-list* $\cup \{\mathtt{refl}\}$
else if $(\hat{\varphi}, \textit{types-list})$ satisfies the transitive case for $k \geq 2$ **then**
types-list $:=$ *types-list* $\cup \{\mathtt{trans}^k\}$
else if $(\hat{\varphi}, \textit{types-list})$ satisfies the symmetric case **then**
types-list $:=$ *types-list* $\cup \{\mathtt{symm}\}$
end if
end if
if for some k, $\{\mathtt{symm}, \mathtt{trans}^k\} \subseteq$ *types-list* **then**
$\mathsf{K}(\hat{\psi})$ has the polynomial-size model property, and $\mathsf{K}(\hat{\psi})$-SAT $\in$ NP.
end if
end while

Figure 3: The Algorithm Horn-Classification

is an edge $(\alpha(y), \alpha(x))$ in T. If one of these cases applies, $\hat{\varphi}$ is satisfied in every reflexive, transitive, or symmetric tree, and hence recalling Theorem 2.3, it is not surprising that these conditions do not lead to NP membership on their own—but in combination with others, they very well might. In the variable *types-list*, the algorithm keeps a list of these conditions encountered.

For the correctness proof, we first need to show that the choices that the algorithm has to make always can be made: In the relevant situations, at least one of the "reflexive," "transitive," or "symmetric" conditions occurs. We also need to prove that it actually comes to a halt—the main argument is to show that only finitely many $\mathtt{trans}^k$-conditions are added to *types-list*, and no element is added twice. Building on Ladner's results, proving correctness of the PSPACE hard cases is trivial. The interesting statement of the classification is that all logics not covered by these cases have an NP-complete satisfiability problem, which is as low a complexity bound as we can hope for. We give an example for a logic which Horn-Classification determines to have a satisfiability problem in NP.

As an example, let $\hat{\varphi}$ be the universal Horn clause with prerequisite graph as shown in Figure 2, with $\mathrm{conc}(\hat{\varphi}) = (x, y)$, and let $\hat{\psi}$ be the Horn formula having $\hat{\varphi}$ as its only clause. Horn-Classification starts with *types-list* $= \emptyset$, and in its first iteration checks if $\hat{\varphi}$ is satisfied in every strict tree. This is not the case, as Figure 4 shows (here, we simply marked each node with the names of the vertices which are preimages of the homomorphism): This is a homomorphic image of $\mathrm{prereq}(\hat{\varphi})$ as a line (in particular a strict tree), in which the images of x and y are not connected with an edge. Thus $\hat{\varphi}$ is not satisfied in this strict

tree. However, if we map $\mathrm{prereq}(\hat{\varphi})$ homomorphically into a strict tree, then the vertices between s and t must be "pairwise identified" like in Figure 4.

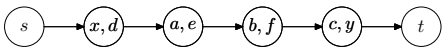


Figure 4: Image as line

Therefore the transitive case is satisfied for $k = 3$, and Horn-Classification adds $\mathtt{trans}^3$ to *types-list*. Next it checks if $\hat{\varphi}$ is satisfied in every $\{\mathtt{trans}^3\}$ tree. This is not the case, and the homomorphic image of $\mathrm{prereq}(\hat{\varphi})$ as a $\{\mathtt{trans}^3\}$ line in Figure 5 (here we only included those edges added by the $\mathtt{trans}^3$ closure required for the homomorphism) shows that $\hat{\varphi}$ satisfies the homomorphic property in $\mathcal{H}_{\mathrm{NP}}$ mentioned at the end of Section 4.2. This also clarifies what a "generalized line" is: At this point, the variable *types-list* only contains the condition $\mathtt{trans}^3$, and hence a "generalized line" is the 3 transitive closure of a line. Therefore, $\mathsf{K}(\hat{\varphi})$-$\mathsf{SAT} \in \mathrm{NP}$.

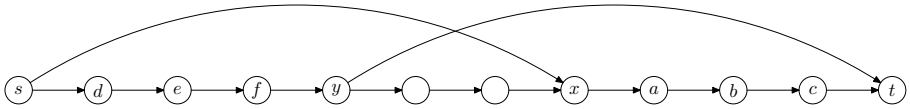


Figure 5: Homomorphic image as $\{\mathtt{trans}^3\}$-line

This example demonstrates that in the run of the algorithm, a clause $\hat{\varphi}$ can meet different cases depending on the content of the variable *types-list* : In the situation that *types-list* $= \emptyset$, the clause $\hat{\varphi}$ satisfies the transitive case, but when *types-list* $= \{\mathtt{trans}^3\}$, this is no longer true.

5. Tree-like Models and PSPACE Upper Bounds

We now prove PSPACE upper bounds for many of our logics. The first step is to prove a tree-like model property for the PSPACE-hard cases. This generalizes many standard results, like the fact that $\mathsf{K4}$-satisfiable formulas are always satisfiable on a transitive tree. For a model M, let $\mathsf{edges}(M)$ denote the edges of the frame that M is based on.

Theorem 5.1. *Let $\hat{\psi}$ be a universal Horn formula satisfied on every types-list tree, where there is no k such that types-list contains both $\mathtt{trans}^k$ and $\mathtt{symm}$. Then every $\mathsf{K}(\hat{\psi})$-satisfiable modal formula is satisfiable in a model M such that M satisfies $\hat{\psi}$ and there is a strict tree T_0 and a types-list tree T_1 such that T_0, T_1, and M have the same set of worlds, and $\mathsf{edges}(T_0) \subseteq \mathsf{edges}(M) \subseteq \mathsf{edges}(T_1)$.*

The theorem shows that in a certain way, the "converse" of the prerequisite that $\hat{\psi}$ is satisfied in every *types-list* tree is true as well: Not only is every *types-list* tree a model in the logic $\mathsf{K}(\hat{\psi})$, but we can restrict ourselves to models which are "close" to *types-list* trees. This is not a real "converse:" For example, if *types-list* contains only the symmetric condition, there might be formulas satisfiable in $\mathsf{K}(\hat{\psi})$, but not in a symmetric tree. The main idea of the proof is to start with a tree model for a satisfiable formula ϕ, and step by step add enough edges to the model in a way which again gives a model for ϕ. Using this theorem, we can construct a PSPACE decision algorithm for the involved logics similar in spirit to Ladner's decision procedure, with the additional difficulty that we need to ensure that the properties demanded by the first-order formula are also met by the model. This is a major obstacle: Recalling Proposition 4.1, we need to consider all homomorphisms from the prerequisite graphs of the clauses in the formula $\hat{\psi}$ into the (potentially exponential size) model that we construct, at the same time may only keep a polynomial fragment of the

model in memory. This can be done since we can restrict ourselves to connected components of small diameter in a strict tree. The theorem gives a unified proof for showing that the logics K, T, and B, among others, have satisfiability problems in PSPACE.

Theorem 5.2. *Let $\hat{\psi}$ be a universal Horn formula such that* Horn-Classification *does not add any* $\mathtt{trans}^k$ *to types-list on input $\hat{\psi}$. Then $\mathsf{K}(\hat{\psi})$-SAT $\in$ PSPACE.*

There are some interesting applications of our results: Ladner proved that all normal modal logics KL such that $\mathsf{S4}$ is an extension of KL have a PSPACE-hard satisfiability problem. Using Theorems 3.2 and 5.1, we show that his result is optimal in the sense that every universal Horn logic which is a "proper extension" of $\mathsf{S4}$ already has an NP-solvable satisfiability problem.

Theorem 5.3. *Let $\hat{\psi}$ be a universal Horn formula such that $\hat{\psi}$ implies $\hat{\varphi}_{\text{refl}} \wedge \hat{\varphi}_{\text{trans}}$. Then either $\mathsf{K}(\hat{\psi}) = \mathsf{S4}$, or $\mathsf{K}(\hat{\psi})$ has the polynomial-size model property and $\mathsf{K}(\hat{\psi})$-SAT $\in$ NP.*

We further can show a PSPACE upper bound for all universal Horn logics which are extensions of the logic T, and hence, from Theorem 3.2, conclude that these are all either solvable in NP (and thus NP-complete if they are consistent), or PSPACE-complete.

Theorem 5.4. *Let $\hat{\psi}$ be a universal Horn formula such that $\hat{\psi}$ implies $\hat{\varphi}_{\text{refl}}$. Then $\mathsf{K}(\hat{\psi})$-SAT $\in$ PSPACE.*

In a similar way, we can prove that all universal Horn logics which imply a variant of symmetry give rise to a satisfiability problem in PSPACE. A noteworthy difference in the prerequisites of Theorem 5.4 and Corollary 5.5 is that the former requires the reflexivity condition to be implied by the formula $\hat{\psi}$, while the latter only needs a "near symmetry"-condition as detected by Horn-Classification.

Corollary 5.5. *Let $\hat{\psi}$ be a universal Horn formula such that* Horn-Classification *adds* $\mathtt{symm}$ *to types-list on input $\hat{\psi}$. Then $\mathsf{K}(\hat{\psi})$-SAT $\in$ PSPACE. In particular, any universal Horn logic which is an extension of B has a satisfiability problem solvable in* PSPACE.

6. Conclusion and Future Research

We analyzed the complexity of modal logics defined by universal Horn formulas, covering many well-known logics. We showed that the non-trivial satisfiability problems for these logics are either NP-complete or PSPACE-hard, and gave an easy criterion to recognize these cases. Our results directly imply that (unless NP = PSPACE) such a logic has a satisfiability problem in NP if and only if it has the polynomial-size model property. We also demonstrated that a wide class of the considered logics has a satisfiability problem solvable in PSPACE.

Open questions include determining complexity upper bounds for the satisfiability problems for all modal logics defined by universal Horn formulas. We strongly conjecture that all of these are decidable, and consider it possible that all of these problems are in PSPACE. A successful way to establish upper complexity bounds is the guarded fragment [AvBN98, Grä99]. This does not seem to be applicable to our logics, since it cannot be used for transitive logics, and we obtain PSPACE-upper bounds for all of our logics except those involving a variant of transitivity.

The next major open challenges are generalizing our results to formulas not in the Horn class, and allowing arbitrary quantification. Initial results show that even when considering

only universal formulas over the frame language, undecidable logics appear. An interesting enrichment of Horn clauses is to allow the equality relation. Preliminary results indicate that Theorem 3.2 holds for this more general case as well.

Acknowledgments: The second author thanks Thomas Schneider for helpful hints. We also thank the anonymous referees for many helpful suggestions.

References

[AvBN98] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

[BG04] B. Bennett and A. Galton. A unifying semantics for time and events. *Artificial Intelligence*, 153(1-2):13–48, 2004.

[BHSS06] M. Bauland, E. Hemaspaandra, H. Schnoor, and I. Schnoor. Generalized modal satisfiability. In *Proc. of the 23nd Symposium on Theoretical Aspects of Computer Science*, pp. 500–511, 2006.

[BZ05] C. Baral and Y. Zhang. Knowledge updates: Semantics and complexity issues. *Artificial Intelligence*, 164(1-2):209–243, 2005.

[CDF03] T. Coffey, R. Dojen, and T. Flanagan. On the automated implementation of modal logics used to verify security protocols. In *Proceedings of the 1st international symposium on Information and communication technologies*, pages 329–334. Trinity College Dublin, 2003.

[CL94] C. Chen and I. Lin. The computational complexity of the satisfiability of modal horn clauses for modal propositional logics. *Theoretical Computer Science*, 129(1):95–121, 1994.

[FHJ02] U. Frendrup, Hüttel, and J. Jensen. Modal logics for cryptographic processes. In *Proceedings of EXPRESS*, 2002.

[Grä99] E. Grädel. Why are modal logics so robustly decidable? *Bulletin of the European Association for Theoretical Computer Science*, 68:90103, 1999.

[Hal95] J. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(2):361–372, 1995.

[HM92] J. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(2):319–379, 1992.

[HMT88] J. Halpern, Y. Moses, and M. Tuttle. A knowledge-based analysis of zero knowledge. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 132–147, New York, NY, USA, 1988. ACM Press.

[HR07] J. Halpern and L. Rêgo. Characterizing the NP-PSPACE gap in the satisfiability problem for modal logic. In *International Joint Conferences on Artificial Intelligence*, pages 2306–2311, 2007.

[Lad77] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977.

[LR86] R. Ladner and J. Reif. The logic of distributed protocols: Preliminary report. In *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 207–222, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc.

[Ngu05] L. Nguyen. On the complexity of fragments of modal logics. In *Advances in Modal Logic - Volume 5*, pages 249–268. King's College Publications, 2005.

[Sah73] H. Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logic. In *Proceedings of the Third Scandinavian Logic Symposium*, 1973.

[SC85] A. Sistla and E. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[SP06] L. Schröder and D. Pattinson. PSPACE bounds for rank-1 modal logics. In *LICS*, pages 231–242, 2006.

[Spa93] E. Spaan. *Complexity of Modal Logics*. PhD thesis, Department of Mathematics and Computer Science, University of Amsterdam, 1993.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 361-372
www.stacs-conf.org

FIXED PARAMETER POLYNOMIAL TIME ALGORITHMS FOR MAXIMUM AGREEMENT AND COMPATIBLE SUPERTREES

VIET TUNG HOANG [1,2] AND WING-KIN SUNG [1,2]

[1] Department of Computer Science, National University of Singapore
E-mail address: {hoangvi2,ksung}@comp.nus.edu.sg

[2] Genome Institute of Singapore

Abstract. Consider a set of labels L and a set of trees $\mathcal{T} = \{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \ldots, \mathcal{T}^{(k)}\}$ where each tree $\mathcal{T}^{(i)}$ is distinctly leaf-labeled by some subset of L. One fundamental problem is to find the biggest tree (denoted as supertree) to represent $\mathcal{T}$ which minimizes the disagreements with the trees in $\mathcal{T}$ under certain criteria. This problem finds applications in phylogenetics, database, and data mining. In this paper, we focus on two particular supertree problems, namely, the maximum agreement supertree problem (MASP) and the maximum compatible supertree problem (MCSP). These two problems are known to be NP-hard for $k \geq 3$. This paper gives the first polynomial time algorithms for both MASP and MCSP when both k and the maximum degree D of the trees are constant.

1. Introduction

Given a set of labels L and a set of unordered trees $\mathcal{T} = \{\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(k)}\}$ where each tree $\mathcal{T}^{(i)}$ is distinctly leaf-labeled by some subset of L. The supertree method tries to find a tree to represent all trees in $\mathcal{T}$ which minimizes the possible conflicts in the input trees. The supertree method finds applications in phylogenetics, database, and data mining. For instance, in the Tree of Life project [10], the supertree method is the basic tool to infer the phylogenetic tree of all species.

Many supertree methods have been proposed in the literature [2, 5, 6, 8]. This paper focuses on two particular supertree methods, namely the Maximum Agreement Supertree (MASP) [8] and the Maximum Compatible Supertree (MCSP) [2]. Both methods try to find a consensus tree with the largest number of leaves which can represent all the trees in $\mathcal{T}$ under certain criteria. (Please read Section 2 for the formal definition.)

MASP and MCSP are known to be NP-hard as they are the generalization of the Maximum Agreement Subtree problem (MAST) [1, 3, 9] and the Maximum Compatible Subtree problem (MCT) [7, 4] respectively. Jansson et al. [8] proved that MASP remains NP-hard even if every tree is a rooted triplet, i.e., a binary tree of 3 leaves. For $k = 2$, Jansson et al. [8] and Berry and Nicolas [2] proposed a linear time algorithm to transform MASP and MCSP for 2 input trees to MAST and MCT respectively. For $k \geq 3$, positive

1998 ACM Subject Classification: Algorithms, Biological computing.
Key words and phrases: maximum agreement supertree, maximum compatible supertree.

	Rooted		**Unrooted**	
MASP for k trees of max degree D	$O((kD)^{kD+3}(2n)^k)$	†	$O((kD)^{kD+3}(4n)^k)$	†
MCSP for k trees of max degree D	$O(2^{2kD}n^k)$	†	$O(2^{2kD}n^k)$	†
MASP/MCSP for k binary trees	$O\big(k(2n^2)^{3k^2}\big)$	[8]		
	$O(8^k n^k)$	[6]		
	$O(6^k n^k)$	†		

Table 1: Summary of previous and new results († stands for new result).

results for computing MASP/MCSP are reported only for rooted binary trees. Jansson et al. [8] gave an $O\big(k(2n)^{3k^2}\big)$ time solution to this problem. Recently, Guillemot and Berry [6] further improve the running time to $O(8^k n^k)$.

In general, the trees in $\mathcal{T}$ may not be binary nor rooted. Hence, Jansson et al. [8] posted an open problem and asked if MASP can be solved in polynomial time when k and the maximum degree of the trees in $\mathcal{T}$ are constant. This paper gives an affirmative answer to this question. We show that both MASP and MCSP can be solved in polynomial time when $\mathcal{T}$ contains constant number of bounded degree trees. For the special case where the trees in $\mathcal{T}$ are rooted binary trees, we show that both MASP and MCSP can be solved in $O(6^k n^k)$ time, which improves the previous best result. Table 1 summarizes the previous and new results.

The rest of the paper is organized as follows. Section 2 gives the formal definition of the problems. Then, Sections 3 and 4 describe the algorithms for solving MCSP for both rooted and unrooted cases. Finally, Sections 5 and 6 detail the algorithms for solving MASP for both rooted and unrooted cases. Proofs omitted due to space limitation will appear in the full version of this paper.

2. Preliminary

A *phylogenetic tree* is defined as an unordered and distinctly leaf-labeled tree. Given a phylogenetic tree T, the notation $L(T)$ denotes the leaf set of T, and the *size* of T refers to $|L(T)|$. For any label set S, the *restriction* of T to S, denoted $T|S$, is a phylogenetic tree obtained from T by removing all leaves in $L(T) - S$ and then suppressing all internal nodes of degree two. (See Figure 1 for an example of *restriction*.) For two phylogenetic trees T and T', we say that T *refines* T', denoted $T \trianglerighteq T'$, if T' can be obtained by contracting some edges of T. (See Figure 1 for an example of *refinement*.)

Maximum Compatible Supertree Problem: Consider a set of k phylogenetic trees $\mathcal{T} = \{\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(k)}\}$. A *compatible supertree* of $\mathcal{T}$ is a tree Y such that $Y|L(\mathcal{T}^{(i)}) \trianglerighteq \mathcal{T}^{(i)}|L(Y)$ for all $i \leq k$. The Maximum Compatible Supertree Problem (MCSP) is to find a compatible supertree with as many leaves as possible. Figure 2 shows an example of a compatible supertree Y of two rooted phylogenetic trees $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$. If all input trees have the same leaf sets, MCSP is referred as Maximum Compatible Subtree Problem (MCT).

Maximum Agreement Supertree Problem: Consider a set of k phylogenetic trees $\mathcal{T} = \{\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(k)}\}$. An *agreement supertree* of $\mathcal{T}$ is a tree X such that $X|L(\mathcal{T}^{(i)}) = \mathcal{T}^{(i)}|L(X)$ for all $i \leq k$. The Maximum Agreement Supertree Problem (MASP) is to find an agreement supertree with as many leaves as possible. Figure 2 shows an example of an

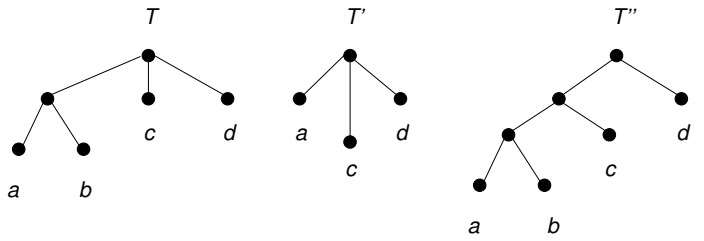


Figure 1: Three rooted trees. A tree T, a tree T' such that $T' = T \mid \{a, c, d\}$, and a tree T'' such that $T'' \trianglerighteq T$.

agreement supertree X of two rooted phylogenetic trees $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$. If all input trees have the same leaf sets, MASP is referred as Maximum Agreement Subtree Problem (MAST).

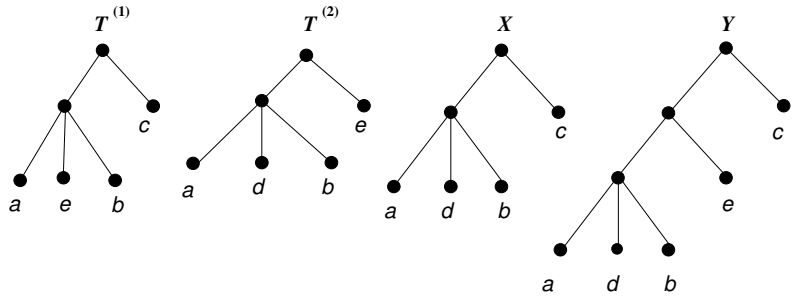


Figure 2: An agreement supertree X and a compatible supertree Y of 2 rooted phylogenetic trees $\mathcal{T}^{(1)}$ and $\mathcal{T}^{(2)}$.

In the following discussion, for the set of phylogenetic trees $\mathcal{T} = \{\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(k)}\}$, we denote $n = |\bigcup_{i=1..k} L(\mathcal{T}^{(i)})|$, and D stands for the maximum degree of the trees in $\mathcal{T}$. We assume that none of the trees in $\mathcal{T}$ has an internal node of degree two, so that each tree contains at most $n-1$ internal nodes. (If a tree $\mathcal{T}^{(i)}$ has some internal nodes of degree two, we can replace it by $\mathcal{T}^{(i)} \mid L(\mathcal{T}^{(i)})$ in linear time.)

3. Algorithm for MCSP of rooted trees

Let $\mathcal{T}$ be a set of k rooted phylogenetic trees. This section presents a dynamic programming algorithm to compute the size of a maximum compatible supertree of $\mathcal{T}$ in $O\left(2^{2kD}n^k\right)$ time. The maximum compatible supertree can be obtained in the same asymptotic time bound by backtracking.

For every compatible supertree Y of $\mathcal{T}$, there exists a binary tree that refines Y. This binary tree is also a compatible supertree of $\mathcal{T}$, and is of the same size as Y. Hence in this section, every compatible supertree is implicitly assumed to be binary.

Definition 3.1 (Cut-subtree)**.** A *cut-subtree* of a tree T is either an empty tree or a tree obtained by first selecting some subtrees attached to the same internal node in T and then connecting those subtrees by a common root.

Definition 3.2 (Cut-subforest)**.** Given a set of k rooted (or unrooted) trees $\mathcal{T}$, a *cut-subforest* of $\mathcal{T}$ is a set $\mathcal{A} = \{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(k)}\}$, where $\mathcal{A}^{(i)}$ is a cut-subtree of $\mathcal{T}^{(i)}$ and at least one element of $\mathcal{A}$ is not an empty tree.

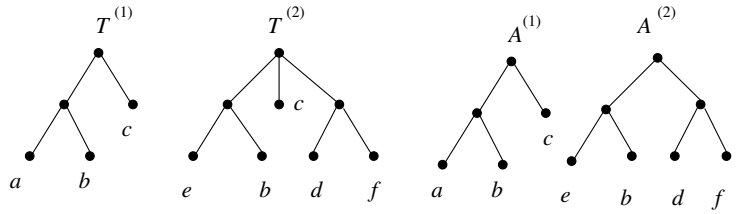


Figure 3: A cut-subforest $\mathcal{A}$ of $\mathcal{T}$.

For example, in Figure 3, $\{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}\}$ is a cut-subforest of $\{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}\}$. Let $\mathcal{O}$ denote the set of all possible cut-subforests of $\mathcal{T}$.

Lemma 3.3. *There are* $O\left(2^{kD}n^k\right)$ *different cut-subforests of* $\mathcal{T}$.

Proof. We claim that each tree $\mathcal{T}^{(i)}$ contributes $2^D n$ or fewer cut-subtrees; therefore there are $O\left(2^{kD}n^k\right)$ cut-subforests of $\mathcal{T}$. At each internal node v of $\mathcal{T}^{(i)}$, since the degree of v does not exceed D, we have at most 2^D ways of selecting the subtrees attached to v to form a cut-subtree. Including the empty tree, the number of cut-subtrees in $\mathcal{T}^{(i)}$ cannot go beyond $(n-1)2^D + 1 < 2^D n$. ∎

Figure 4 demonstrates that a compatible supertree of some cut-subforest $\mathcal{A}$ of $\mathcal{T}$ may not be a compatible supertree of $\mathcal{T}$. To circumvent this irregularity, we define *embedded supertree* as follows.

Definition 3.4 (Embedded supertree)**.** For any cut-subforest $\mathcal{A}$ of $\mathcal{T}$, a tree Y is called an *embedded supertree* of $\mathcal{A}$ if Y is a compatible supertree of $\mathcal{A}$, and $L(Y) \cap L(\mathcal{T}^{(i)}) \subseteq L(\mathcal{A}^{(i)})$ for all $i \leq k$.

Note that a compatible supertree of $\mathcal{T}$ is also an embedded supertree of $\mathcal{T}$. For each cut-subforest $\mathcal{A}$ of $\mathcal{T}$, let $\texttt{mcsp}(\mathcal{A})$ denote the maximum size of embedded supertrees of $\mathcal{A}$. Our aim is to compute $\texttt{mcsp}(\mathcal{T})$. Below, we first define the recursive equation for computing $\texttt{mcsp}(\mathcal{A})$ for all cut-subforests $\mathcal{A} \in \mathcal{O}$. Then, we describe our dynamic programming algorithm.

We partition the cut-subforests in $\mathcal{O}$ into two classes. A cut-subforest $\mathcal{A}$ of $\mathcal{T}$ is *terminal* if each element $\mathcal{A}^{(i)}$ is either an empty tree or a leaf of $\mathcal{T}^{(i)}$; it is called *non-terminal*, otherwise.

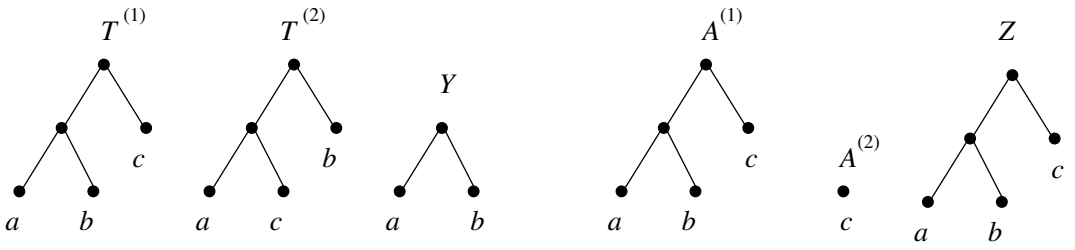


Figure 4: Consider $\mathcal{T} = \{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}\}$ and its cut-subforest $\mathcal{A} = \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}\}$. Although Z is a compatible supertree of $\mathcal{A}$, it is not a compatible supertree of $\mathcal{T}$. The maximum compatible supertree of $\mathcal{T}$ is Y that contains only 2 leaves.

For each terminal cut-subforest $\mathcal{A}$, let

$$\Lambda(\mathcal{A}) = \Big\{ l \in \bigcup_{j=1..k} L(\mathcal{A}^{(j)}) \mid l \notin L(\mathcal{T}^{(i)}) - L(\mathcal{A}^{(i)}) \text{ for } i = 1, 2, \ldots, k \Big\} \ . \tag{3.1}$$

For example, with $\mathcal{T}$ in Figure 2, if $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ are leaves labeled by a and d respectively then $\Lambda(\mathcal{A}) = \{d\}$. In Lemma 3.5, we show that $\texttt{mcsp}(\mathcal{A}) = |\Lambda(\mathcal{A})|$.

Lemma 3.5. *If $\mathcal{A}$ is a terminal cut-subforest then* $\texttt{mcsp}(\mathcal{A}) = |\Lambda(\mathcal{A})|$.

Proof. Consider any embedded supertree Y of $\mathcal{A}$. By Definition 3.4, every leaf of Y belongs to $\Lambda(\mathcal{A})$. Hence the value $\texttt{mcsp}(\mathcal{A})$ does not exceed $|\Lambda(\mathcal{A})|$.

It remains to give an example of some embedded supertree of $\mathcal{A}$ whose leaf set is $\Lambda(\mathcal{A})$. Let C be a rooted caterpillar [1] whose leaf set is $\Lambda(\mathcal{A})$. The definition of $\Lambda(\mathcal{A})$ implies that $L(C) \cap L(\mathcal{T}^{(i)}) \subseteq L\big(\mathcal{A}^{(i)}\big)$ for every $i \leq k$. Since each $\mathcal{A}^{(i)}$ has at most one leaf, it is straightforward that C is a compatible supertree of $\mathcal{A}$. Hence C is the desired example. ∎

Definition 3.6 (Bipartite)**.** Let $\mathcal{A}$ be a cut-subforest of $\mathcal{T}$. We say that the cut-subforests $\mathcal{A}_L$ and $\mathcal{A}_R$ *bipartition* $\mathcal{A}$ if for every $i \leq k$, the trees $\mathcal{A}_L^{(i)}$ and $\mathcal{A}_R^{(i)}$ can be obtained by (1) partitioning the subtrees attached to the root of $\mathcal{A}^{(i)}$ into two sets $S_L^{(i)}$ and $S_R^{(i)}$; and (2) connecting the subtrees in $S_L^{(i)}$ (resp. $S_R^{(i)}$) by a common root to form $\mathcal{A}_L^{(i)}$ (resp. $\mathcal{A}_R^{(i)}$).

Figure 5 shows an example of the preceding definition. For each non-terminal cut-subforest $\mathcal{A}$, we compute $\texttt{mcsp}(\mathcal{A})$ based on the mcsp values of $\mathcal{A}_L$ and $\mathcal{A}_R$ for each bipartite $(\mathcal{A}_L, \mathcal{A}_R)$ of $\mathcal{A}$. More precisely, we prove that

$$\texttt{mcsp}(\mathcal{A}) = \max\{\texttt{mcsp}(\mathcal{A}_L) + \texttt{mcsp}(\mathcal{A}_R) \mid \mathcal{A}_L \text{ and } \mathcal{A}_R \text{ bipartition } \mathcal{A}\} \ . \tag{3.2}$$

The identity (3.2) is then established by Lemmas 3.8 and 3.10.

Lemma 3.7. *Consider a bipartite $(\mathcal{A}_L, \mathcal{A}_R)$ of some cut-subforest $\mathcal{A}$ of $\mathcal{T}$. If Y_L and Y_R are embedded supertrees of $\mathcal{A}_L$ and $\mathcal{A}_R$ respectively then Y is an embedded supertree of $\mathcal{A}$, where Y is formed by connecting Y_L and Y_R to a common root.*

[1] A rooted caterpillar is a rooted, unordered, and distinctly leaf-labeled binary tree where every internal node has at least one child that is a leaf.

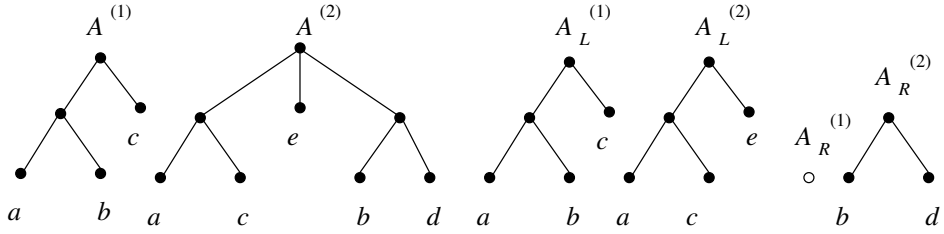


Figure 5: A bipartite $(\mathcal{A}_L, \mathcal{A}_R)$ of a cut-subforest $\mathcal{A}$. The empty tree is represented by a white circle.

Lemma 3.8. *Let $\mathcal{A}$ be a cut-subforest of $\mathcal{T}$. If $(\mathcal{A}_L, \mathcal{A}_R)$ is a bipartite of $\mathcal{A}$ then* $\mathtt{mcsp}(\mathcal{A}) \geq \mathtt{mcsp}(\mathcal{A}_L) + \mathtt{mcsp}(\mathcal{A}_R)$.

Proof. Consider an embedded supertree Y_L of $\mathcal{A}_L$ such that $|L(Y_L)| = \mathtt{mcsp}(\mathcal{A}_L)$. Define Y_R for $\mathcal{A}_R$ similarly. Let Y be a tree formed by connecting Y_L and Y_R with a common root. Note that Y is of size $\mathtt{mcsp}(\mathcal{A}_L) + \mathtt{mcsp}(\mathcal{A}_R)$. By Lemma 3.7, Y is an embedded supertree of $\mathcal{A}$ and hence the lemma follows. ∎

Lemma 3.9. *Given a cut-subforest $\mathcal{A}$ of $\mathcal{T}$, let Y be a binary embedded supertree of $\mathcal{A}$ with left subtree Y_L and right subtree Y_R. There exists a bipartite $(\mathcal{A}_L, \mathcal{A}_R)$ of $\mathcal{A}$ such that either (i) Y is an embedded supertree of $\mathcal{A}_L$; or (ii) Y_L and Y_R are embedded supertrees of $\mathcal{A}_L$ and $\mathcal{A}_R$ respectively.*

Lemma 3.10. *For each non-terminal cut-subforest $\mathcal{A}$ of $\mathcal{T}$, there exists a bipartite $(\mathcal{A}_L, \mathcal{A}_R)$ of $\mathcal{A}$ such that* $\mathtt{mcsp}(\mathcal{A}) \leq \mathtt{mcsp}(\mathcal{A}_L) + \mathtt{mcsp}(\mathcal{A}_R)$.

Proof. Let Y be a binary embedded supertree of $\mathcal{A}$ such that $|L(Y)| = \mathtt{mcsp}(\mathcal{A})$. By Lemma 3.9, there exists a bipartite $(\mathcal{A}_L, \mathcal{A}_R)$ of $\mathcal{A}$ such that either (1) Y is an embedded supertree of $\mathcal{A}_L$; or (2) Y_L and Y_R are embedded supertrees of $\mathcal{A}_L$ and $\mathcal{A}_R$ respectively, where Y_L is the left subtree and Y_R is the right subtree of Y. In both cases, $|L(Y)| \leq \mathtt{mcsp}(\mathcal{A}_L) + \mathtt{mcsp}(\mathcal{A}_R)$. Then the lemma follows. ∎

The above discussion then leads to Theorem 3.11.

Theorem 3.11. *For every cut-subforest $\mathcal{A}$ of $\mathcal{T}$, the value* $\mathtt{mcsp}(\mathcal{A})$ *equals to*

$$\begin{cases} |\Lambda(\mathcal{A})|\,, & \text{if } \mathcal{A} \text{ is terminal,} \\ \max\{\mathtt{mcsp}(\mathcal{A}_L) + \mathtt{mcsp}(\mathcal{A}_R) \mid \mathcal{A}_L \text{ and } \mathcal{A}_R \text{ bipartition } \mathcal{A}\}, & \text{otherwise}\ . \end{cases}$$

We define an ordering of the cut-subforests in $\mathcal{O}$ as follows. For any cut-subforests $\mathcal{A}_1, \mathcal{A}_2$ in $\mathcal{O}$, we say that $\mathcal{A}_1$ is smaller than $\mathcal{A}_2$ if $\mathcal{A}_1^{(i)}$ is a cut-subtree of $\mathcal{A}_2^{(i)}$ for $i = 1, 2, \ldots, k$. Our algorithm enumerates $\mathcal{A} \in \mathcal{O}$ in topologically increasing order and computes $\mathtt{mcsp}(\mathcal{A})$ based on Theorem 3.11. Theorem 3.12 states the complexity of our algorithm.

Theorem 3.12. *A maximum compatible supertree of k rooted phylogenetic trees can be obtained in $O\left(2^{2kD} n^k\right)$ time .*

Proof. Testing if a cut-subforest is terminal takes $O(k)$ times, and each terminal cut-subforest $\mathcal{A}$ then requires $O(k^2)$ time for the computation of $\Lambda(\mathcal{A})$. In view of Lemma 3.3, it suffices to show that each non-terminal cut-subforest $\mathcal{A}$ has $O(2^{kD})$ bipartites. This result follows from the fact that for each $i \leq k$, there are at most 2^D ways to partition the set of the subtrees attached to the root of $\mathcal{A}^{(i)}$. ∎

In the special case where every tree $\mathcal{T}^{(i)}$ is binary, Theorem 3.13 shows that our algorithm actually has a better time complexity. Note that the concepts of agreement supertree and compatible supertree will coincide for binary trees. Hence, our algorithm improves the $O\left(8^k n^k\right)$-time algorithm in [6] for computing maximum agreement supertree of k rooted binary trees.

Theorem 3.13. *If every tree in $\mathcal{T}$ is binary, a maximum compatible supertree (or a maximum agreement supertree) can be computed in $O\left(6^k n^k\right)$ time.*

Proof. We claim that the processing of non-terminal cut-subforests of $\mathcal{T}$ requires $O\left(6^k n^k\right)$ time. The argument in the proof of Theorem 3.12 tells that the remaining computation runs within the same asymptotic time bound. Consider an integer $r \in \{0, 1, \ldots, k\}$. We shall be dealing with a cut-subforest $\mathcal{A}$ such that there are exactly r cut-subtrees $\mathcal{A}^{(i)}$ whose roots are internal nodes of $\mathcal{T}^{(i)}$. The key of this proof is to show that the number of those cut-subforests does not exceed $\left(\begin{array}{c} k \\ r \end{array} \right) (n-1)^r (n+1)^{k-r}$, and the running time for each cut-subforest is $O\left(4^r 2^{k-r}\right)$. Hence, the total running time for all non-terminal cut-subforests is

$$\sum_{r=0}^{k} \left(\begin{array}{c} k \\ r \end{array} \right) (n-1)^r (n+1)^{k-r} O\left(4^r 2^{k-r}\right) = O\left(6^k n^k\right).$$

We can count the number of the specified cut-subforests $\mathcal{A}$ as follows. First there are $\left(\begin{array}{c} k \\ r \end{array} \right)$ options for r indices i such that the roots of cut-subtrees $\mathcal{A}^{(i)}$ are internal nodes of $\mathcal{T}^{(i)}$. For those cut-subtrees, we then appoint one of the $(n-1)$ or fewer internal nodes of $\mathcal{T}^{(i)}$ to be the root node of $\mathcal{A}^{(i)}$. Every other cut-subtree of $\mathcal{A}$ is a leaf or the empty tree, and then can be determined from at most $n+1$ alternatives. Multiplying those possibilities gives us the bound stipulated in the preceding paragraph.

It remains to estimate the running time for each specified cut-subforest $\mathcal{A}$. This task requires us to bound the number of bipartites of each cut-subforest. If the root v of $\mathcal{A}^{(i)}$ is an internal node of $\mathcal{T}^{(i)}$ then $\mathcal{A}^{(i)}$ contributes 4 or fewer ways of partitioning the set of the subtrees attached to v. Otherwise, we have at most 2 ways of partitioning this set. Hence $\mathcal{A}$ owns at most $4^r 2^{k-r}$ bipartites, and this completes the proof. ∎

4. Algorithm for MCSP of unrooted trees

Let $\mathcal{T}$ be a set of k unrooted phylogenetic trees. This section extends the algorithm in Section 3 to find the size of a maximum compatible supertree of $\mathcal{T}$. The maximum compatible supertree can be obtained by backtracking. Surprisingly, the extended algorithm for unrooted trees runs within the same asymptotic time bound as the original algorithm for rooted trees.

We will follow the same approach as Section 3, i.e., for each cut-subforest $\mathcal{A}$ of $\mathcal{T}$, we find an embedded supertree of $\mathcal{A}$ of maximum size. Definitions 3.1, 3.2, and 3.4 for cut-subforest and embedded supertree in the previous section are still valid for unrooted trees. Notice that although $\mathcal{T}$ is the set of unrooted trees, each cut-subforest $\mathcal{A}$ of $\mathcal{T}$ consists of rooted trees. (See Figure 6 for an example of cut-subforest for unrooted trees.) Hence we can use the algorithm in Section 3 to find the maximum embedded supertree of $\mathcal{A}$. We then select the biggest tree T among those maximum embedded supertrees for all cut-subforests of $\mathcal{T}$, and unroot T to obtain the maximum compatible supertree of $\mathcal{T}$.

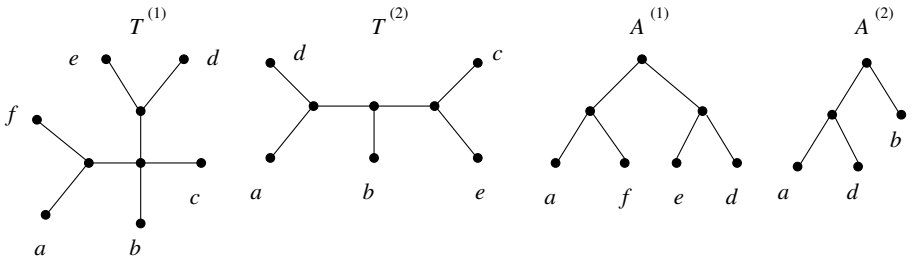


Figure 6: The set of rooted trees $\mathcal{A} = \{\mathcal{A}^{(1)}, \mathcal{A}^{(2)}\}$ is a cut-subforest of $\mathcal{T} = \{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}\}$.

Theorem 4.1 shows that the extended algorithm has the same asymptotic time bound as the algorithm in Section 3.

Theorem 4.1. *We can find a maximum compatible supertree of k unrooted phylogenetic trees in $O\left(2^{2kD}n^k\right)$ time.*

Proof. Using a similar proof as Lemma 3.3, we can prove that there are $O\left(2^{kD}n^k\right)$ cut-subforests of $\mathcal{T}$. As given in the proof of Theorem 3.12, finding the maximum embedded supertrees of each cut-subforest takes $O(2^{kD})$ time. Hence the extended algorithm runs within the specified time bound. ∎

5. Algorithm for MASP of rooted trees

Let $\mathcal{T}$ be a set of k rooted phylogenetic trees. This section presents a dynamic programming algorithm to compute the size of a maximum agreement supertree of $\mathcal{T}$ in $O\left((kD)^{kD+3}(2n)^k\right)$ time. The maximum agreement supertree can be obtained in the same asymptotic time bound by backtracking.

The idea here is similar to that of Section 3. However, while we can assume that compatible supertrees are binary, the maximum degree of agreement supertrees can grow up to kD. It is the reason why we have the factor $O((kD)^{kD+3})$ in the complexity.

Definition 5.1 (Sub-forest)**.** Given a set of k rooted trees $\mathcal{T}$, a *sub-forest* of $\mathcal{T}$ is a set $\mathcal{A} = \{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(k)}\}$, where each $\mathcal{A}^{(i)}$ is either an empty tree or a complete subtree rooted at some node of $\mathcal{T}^{(i)}$, and at least one element of $\mathcal{A}$ is not an empty tree.

Notice that the definition of sub-forest does not coincide with the concept of cut-subforest in Definition 3.2 of Section 3. For example, the cut-subforest $\mathcal{A}$ in Figure 3 is not a sub-forest of $\mathcal{T}$, because $\mathcal{A}^{(2)}$ is not a complete subtree rooted at some node of $\mathcal{T}^{(2)}$. Let $\mathcal{O}$ denote the set of all possible sub-forests of $\mathcal{T}$. Then $|\mathcal{O}| = O\left((2n)^k\right)$.

Definition 5.2 (Enclosed supertree)**.** For any sub-forest $\mathcal{A}$ of $\mathcal{T}$, a tree X is called an *enclosed supertree* of $\mathcal{A}$ if X is an agreement supertree of $\mathcal{A}$, and $L(X) \cap L(\mathcal{T}^{(i)}) \subseteq L(\mathcal{A}^{(i)})$ for all $i \leq k$.

For each sub-forest $\mathcal{A}$ of $\mathcal{T}$, let $\mathtt{masp}(\mathcal{A})$ denote the maximum size of enclosed supertrees of $\mathcal{A}$. We use a similar approach as Section 3, i.e., we compute $\mathtt{masp}(\mathcal{A})$ for all $\mathcal{A} \in \mathcal{O}$, and $\mathtt{masp}(\mathcal{T})$ is the size of a maximum agreement supertree of $\mathcal{T}$. We partition the sub-forests in $\mathcal{O}$ to two classes. A sub-forest $\mathcal{A}$ is *terminal* if each $\mathcal{A}^{(i)}$ is either an empty tree or a leaf. Otherwise, $\mathcal{A}$ is called *non-terminal*.

Notice that for terminal sub-forest, the definition of enclosed supertree coincides with the concept of embedded supertree in Definition 3.4 of Section 3. Then by Lemma 3.5, we have $\mathtt{masp}(\mathcal{A}) = |\Lambda(\mathcal{A})|$. (Please refer to the formula (3.1) in the paragraph preceding Lemma 3.5 for the definition of function Λ.)

Definition 5.3 (Decomposition)**.** Let $\mathcal{A}$ be a sub-forest of $\mathcal{T}$. We say that sub-forests $\mathcal{B}_1, \ldots, \mathcal{B}_d$ (with $d \geq 2$) *decompose* $\mathcal{A}$ if for all $i \leq k$, either (i) Exactly one of $\mathcal{B}_1^{(i)}, \ldots, \mathcal{B}_d^{(i)}$ is isomorphic to $\mathcal{A}^{(i)}$ while the others are empty trees; or (ii) There are at least 2 nonempty trees in $\mathcal{B}_1^{(i)}, \ldots, \mathcal{B}_d^{(i)}$, and all those nonempty trees are isomorphic to pairwise distinct subtrees attached to the root of $\mathcal{A}^{(i)}$.

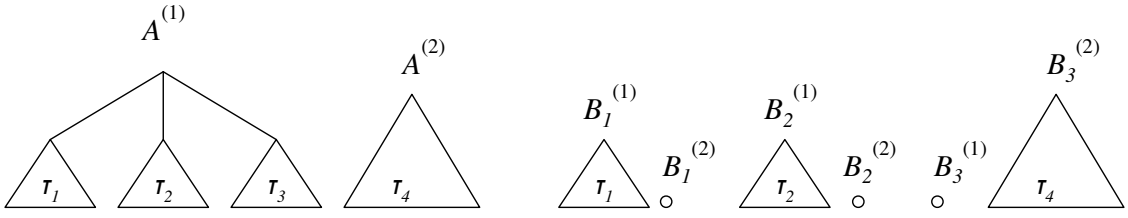


Figure 7: A decomposition $(\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3)$ of a sub-forest $\mathcal{A}$. The empty trees are represented by white circles.

Figure 7 illustrates the concept of decomposition. For each sub-forest $\mathcal{A}$ of $\mathcal{T}$, we will prove that

$$\mathtt{masp}(\mathcal{A}) = \max\{\mathtt{masp}(\mathcal{B}_1) + \ldots + \mathtt{masp}(\mathcal{B}_d) \mid \mathcal{B}_1, \ldots, \mathcal{B}_d \text{ decompose } \mathcal{A}\} \ . \tag{5.1}$$

The identity (5.1) is then established by Lemmas 5.5 and 5.7.

Lemma 5.4. *Suppose $(\mathcal{B}_1, \ldots, \mathcal{B}_d)$ is a decomposition of some sub-forest $\mathcal{A}$ of $\mathcal{T}$. Let $\tau_1, \ldots, \tau_d$ be some enclosed supertrees of $\mathcal{B}_1, \ldots, \mathcal{B}_d$ respectively, and let X be the tree obtained by connecting $\tau_1, \ldots, \tau_d$ to a common root. Then, X is an enclosed supertree of $\mathcal{A}$.*

Lemma 5.5. *If $(\mathcal{B}_1, \ldots, \mathcal{B}_d)$ is a decomposition of a sub-forest $\mathcal{A}$ of $\mathcal{T}$ then* $\mathtt{masp}(\mathcal{A}) \geq \mathtt{masp}(\mathcal{B}_1) + \ldots + \mathtt{masp}(\mathcal{B}_d)$.

Proof. For each $\mathcal{B}_j$, let τ_j be an enclosed supertree of $\mathcal{B}_j$ such that $|L(\tau_j)| = \mathtt{masp}(\mathcal{B}_j)$. Let X be the tree obtained by connecting $\tau_1, \ldots, \tau_d$ to a common root. By Lemma 5.4, X is an enclosed supertree of $\mathcal{A}$. Hence $|L(\tau_1)| + \ldots + |L(\tau_d)| = |L(X)| \leq \mathtt{masp}(\mathcal{A})$. ∎

Lemma 5.6. *Let X be an enclosed supertree of some sub-forest $\mathcal{A}$ of $\mathcal{T}$, and let $\tau_1, \ldots, \tau_d$ be all subtrees attached to the root of X. Then either (i) There is a decomposition $(\mathcal{B}_1, \mathcal{B}_2)$ of $\mathcal{A}$ such that X is an enclosed supertree of $\mathcal{B}_1$; or (ii) There is a decomposition $(\mathcal{B}_1, \ldots, \mathcal{B}_d)$ of $\mathcal{A}$ such that each τ_j is an enclosed supertree of $\mathcal{B}_j$.*

Lemma 5.7. *For each non-terminal sub-forest $\mathcal{A}$ of $\mathcal{T}$, there is a decomposition $(\mathcal{B}_1, \ldots, \mathcal{B}_d)$ of $\mathcal{A}$ such that $\texttt{masp}(\mathcal{A}) \leq \texttt{masp}(\mathcal{B}_1) + \ldots + \texttt{masp}(\mathcal{B}_d)$*

Proof. Let X be an enclosed supertree of $\mathcal{A}$ such that $|L(X)| = \texttt{masp}(\mathcal{A})$ and let $\tau_1, \ldots, \tau_d$ be all subtrees attached to the root of X. By Lemma 5.6, either (i) There exists a decomposition $(\mathcal{B}_1, \mathcal{B}_2)$ of $\mathcal{A}$ such that X is an enclosed supertree of $\mathcal{B}_1$; or (ii) There is a decomposition $(\mathcal{B}_1, \ldots, \mathcal{B}_d)$ of $\mathcal{A}$ such that each τ_j is an enclosed supertree of $\mathcal{B}_j$. In case (i), we have $|L(X)| \leq \texttt{masp}(\mathcal{B}_1) \leq \texttt{masp}(\mathcal{B}_1) + \texttt{masp}(\mathcal{B}_2)$. On the other hand, in case (ii), we have $|L(X)| = |L(\tau_1)| + \ldots + |L(\tau_d)| \leq \texttt{masp}(\mathcal{B}_1) + \ldots + \texttt{masp}(\mathcal{B}_d)$. ∎

The above discussion then leads to Theorem 5.8.

Theorem 5.8. *For every sub-forest $\mathcal{A}$ of $\mathcal{T}$, the value $\texttt{masp}(\mathcal{A})$ equals to*

$$\begin{cases} |\Lambda(\mathcal{A})|, & \text{if } \mathcal{A} \text{ is terminal,} \\ \max\{\texttt{masp}(\mathcal{B}_1) + \ldots + \texttt{masp}(\mathcal{B}_d) \mid \mathcal{B}_1, \ldots, \mathcal{B}_d \text{ decompose } \mathcal{A}\}, & \text{otherwise .} \end{cases}$$

We define an ordering of the sub-forests in $\mathcal{O}$ as follows. For any sub-forests $\mathcal{A}_1, \mathcal{A}_2$ in $\mathcal{O}$, we say $\mathcal{A}_1$ is smaller than $\mathcal{A}_2$ if $\mathcal{A}_1^{(i)}$ is either an empty tree or a subtree of $\mathcal{A}_2^{(i)}$ for $i = 1, 2, \ldots, k$. Our algorithm enumerates $\mathcal{A} \in \mathcal{O}$ in topologically increasing order and computes $\texttt{masp}(\mathcal{A})$ based on Theorem 5.8.

In Lemma 5.9, we bound the number of decompositions of each sub-forest of $\mathcal{T}$. Theorem 5.10 states the complexity of the algorithm.

Lemma 5.9. *Each sub-forest of $\mathcal{T}$ has $O\left((kD)^{kD+1}\right)$ decompositions, and generating those decompositions takes $O\left(k^2D^2\right)$ time per decomposition.*

Proof. Let $\mathcal{A}$ be a sub-forest of $\mathcal{T}$. Since the maximum degree of any agreement supertree of $\mathcal{A}$ is bounded by kD, we consider only decompositions that consist of at most kD elements. We claim that for each $d \in \{2, \ldots, kD\}$, the sub-forest $\mathcal{A}$ owns $O\left((d+2)^{kD}\right)$ decompositions $(\mathcal{B}_1, \ldots, \mathcal{B}_d)$. Summing up those asymptotic terms gives us the specified bound.

The key of this proof is to prove that for each $s \in \{1, \ldots, k\}$, the tree $\mathcal{A}^{(s)}$ contributes at most $(d+1)^D + d < (d+2)^D$ sequences $\mathcal{B}_1^{(s)}, \ldots, \mathcal{B}_d^{(s)}$, and generating those sequences requires $O(d)$ time per sequence. We have two cases, each corresponds to a type of the above sequence.

Case 1: One term in the sequence is $\mathcal{A}^{(s)}$; therefore the other terms are empty trees. Then, we can generate this sequence by assigning $\mathcal{A}^{(s)}$ to exactly one term and setting the rest to be empty trees. This case provides exactly d sequences and enumerates them in $O(d)$ time per sequence.

Case 2: No term in the above sequence is $\mathcal{A}^{(s)}$. Consider an integer $r \in \{0, 1, \ldots, d\}$ and assume that the sequence consists of exactly r terms that are nonempty nodes. Then those r nonempty trees are isomorphic to pairwise distinct subtrees attached to the root of $\mathcal{A}^{(s)}$. Let δ be the degree of the root of $\mathcal{A}^{(s)}$. We generate the sequence as follows. First we draw r pairwise distinct subtrees attached to the root of $\mathcal{A}^{(s)}$. Next, we select r terms

in the sequence and distribute the above subtrees to them. Finally we set the remaining terms to be empty trees. Hence this case gives at most

$$\sum_{r\leq\min\{\delta,d\}} \left(\begin{array}{c} \delta \\ r \end{array} \right) \frac{d!}{(d-r)!} < \sum_{r=0}^{D} \left(\begin{array}{c} D \\ r \end{array} \right) d^r = (d+1)^D$$

sequences, and generates them in $O(d)$ time per sequence. ∎

Theorem 5.10. *A maximum agreement supertree of k rooted phylogenetic trees can be obtained in $O\left((kD)^{kD+3}(2n)^k\right)$ time.*

Proof. Testing if a sub-forest is terminal takes $O(k)$ times, and each terminal sub-forest $\mathcal{A}$ then requires $O(k^2)$ time for computing $\Lambda(\mathcal{A})$. By Lemma 5.9, each non-terminal sub-forest requires $O\left((kD)^{kD+3}\right)$ running time. Summing up those asymptotic terms for $O\left((2n)^k\right)$ sub-forests of $\mathcal{T}$ gives us the specified time bound. ∎

6. Algorithm for MASP of unrooted trees

Let $\mathcal{T}$ be a set of k unrooted phylogenetic trees. This section extends the algorithm in Section 5 to find the size of a maximum agreement supertree of $\mathcal{T}$ in $O\left((kD)^{kD+3}(4n)^k\right)$ time. The maximum agreement supertree can be obtained by backtracking.

We say that a set of k rooted trees $\mathcal{F} = \{\mathcal{F}^{(1)}, \ldots, \mathcal{F}^{(k)}\}$ is a *rooted variant* of $\mathcal{T}$ if we can obtain each $\mathcal{F}^{(i)}$ by rooting $\mathcal{T}^{(i)}$ at some internal node. One naive approach is to use the algorithm in the previous section to solve MASP for each rooted variant of $\mathcal{T}$. Each rooted variant then gives us a solution, and the maximum of those solutions is the size of a maximum agreement supertree of $\mathcal{T}$. Because there are $O\left(n^k\right)$ rooted variants of $\mathcal{T}$, this approach adds an $O\left(n^k\right)$ factor to the complexity of the algorithm for rooted trees.

We now show how to improve the above naive algorithm. As mentioned in the previous section, the computation of each rooted variant of $\mathcal{T}$ consists of $O\left((2n)^k\right)$ sub-problems which correspond to its sub-forests. (Please refer to Definition 5.1 for the concept of sub-forest.) Since different rooted variants may have some common sub-forests, the total number of sub-problems we have to run is much smaller than $O(2^k n^{2k})$. More precisely, we will show that the total number of sub-problems is only $O\left((4n)^k\right)$.

A (rooted or unrooted) tree is *trivial* if it is a leaf or an empty tree. A *maximal subtree* of an unrooted tree T is a rooted tree obtained by first rooting T at some internal node v and then removing at most one nontrivial subtree attached to v. Let $\mathcal{O}$ denote the set of sub-forests of all rooted variants of $\mathcal{T}$.

Lemma 6.1. *Let $\mathcal{A} = \{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(k)}\}$ be a set of rooted trees. Then $\mathcal{A} \in \mathcal{O}$ if and only if each $\mathcal{A}^{(i)}$ is either a trivial subtree or a maximal subtree of $\mathcal{T}^{(i)}$.*

Proof. Let $\mathcal{F}$ be a rooted variant of $\mathcal{T}$ such that $\mathcal{A}$ is a sub-forest of $\mathcal{F}$. Fix an index $s \in \{1, \ldots, k\}$ and let v be the root node of $\mathcal{A}^{(s)}$. Our claim is straightforward if either $\mathcal{A}^{(s)}$ is trivial or v is the root node of $\mathcal{F}^{(s)}$. Otherwise, let u be the parent of v in $\mathcal{F}^{(s)}$. Hence $\mathcal{A}^{(s)}$ is the maximal subtree of $\mathcal{T}^{(s)}$ obtained by first rooting $\mathcal{T}^{(s)}$ at v and then removing the complete subtree rooted at u.

Conversely, we construct a rooted variant $\mathcal{F}$ of $\mathcal{T}$ such that $\mathcal{A}$ is a sub-forest of $\mathcal{F}$ as follows. For each $i \leq k$, if $\mathcal{A}^{(i)}$ is trivial or $\mathcal{A}^{(i)}$ is a tree obtained by rooting $\mathcal{T}^{(i)}$ at some internal node then constructing $\mathcal{F}^{(i)}$ is straightforward. Otherwise $\mathcal{A}^{(i)}$ is a maximal subtree

of $\mathcal{T}^{(i)}$ obtained by first rooting $\mathcal{T}^{(i)}$ at some internal node v and then removing exactly one nontrivial subtree τ attached to v. Hence $\mathcal{F}^{(i)}$ is the tree obtained by rooting $\mathcal{T}^{(i)}$ at u, where u is the root of τ. ∎

Theorem 6.2. *We can find a maximum agreement supertree of k unrooted phylogenetic trees in $O\left((kD)^{kD+3}(4n)^k\right)$ time.*

Proof. The key of this proof is to show that each tree $\mathcal{T}^{(i)}$ contributes at most $(3n-1)$ maximal subtrees. It follows that $|\mathcal{O}| \leq (4n)^k$. The specified running time of our algorithm is then straightforward because each subproblem requires $O\left((kD)^{kD+3}\right)$ time as given in the proof of Theorem 5.10. Assume that the tree $\mathcal{T}^{(i)}$ has exactly L leaves, with $L \leq n$. We now count the number of maximal subtrees T of $\mathcal{T}^{(i)}$ in two cases.

Case 1: T is obtained by rooting $\mathcal{T}^{(i)}$ at some internal node. Hence this case provides at most $L - 1 < n$ maximal subtrees.

Case 2: T is obtained by first rooting $\mathcal{T}^{(i)}$ at some internal node v and then removing a nontrivial subtree τ attached to v. Notice that there is a one-to-one correspondence between the tree T and the directed edge (v, u) of $\mathcal{T}^{(i)}$, where u is the root node of τ. There are $2L - 2$ or fewer undirected edges in $\mathcal{T}^{(i)}$ but exactly L of them are adjacent to the leaves. Hence this case gives us at most $2(2L - 2 - L) < 2n - 1$ maximal subtrees. ∎

References

[1] A. Amir and D. Keselman. Maximum Agreement Subtree in a set of Evolutionary Trees: Metrics and Efficient Algorithms. *SIAM Journal on Computing*, 26(6):1656–1669, 1997.

[2] V. Berry and F. Nicolas. Maximum Agreement and Compatible Supertrees. In *Proc. 15th Symposium on Combinatorial Pattern Matching* (CPM 2004), *Lect. Notes in Comp. Science* **3109**, pp. 205–219. Springer, 2004.

[3] M. Farach, T. Przytycka, and M. Thorup. On the agreement of many trees. *Information Processing Letters*, 55:297–301, 1995.

[4] G. Ganapathysaravanabavan and T. Warnow. Finding a maximum compatible tree for a bounded number of trees with bounded degree is solvable in polynomial time. In *Proc. 1st Workshop on Algorithms in Bioinformatics* (WABI 2001), *Lect. Notes in Comp. Science* **2149**, pp. 156–163. Springer, 2001.

[5] A. G. Gordon. Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labelled leaves. *Journal of Classification*, 3:335–348, 1986.

[6] Sylvain Guillemot and Vincent Berry. Fixed-Parameter Tractability of the Maximum Agreement Supertree Problem. In *Proc. 18th Symposium on Combinatorial Pattern Matching* (CPM 2007), *Lect. Notes in Comp. Science* **4580**, pp. 274–285. Springer, 2007.

[7] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71:153–169, 1996.

[8] Jesper Jansson, Joseph H.-K. Ng, Kunihiko Sadakane, and Wing-King Sung. Rooted Maximum Agreement Supertrees. *Algorithmica*, 43:293–307, 2005.

[9] M.-Y. Kao, T.-W. Lam, W.-K. Sung, and H.-F. Ting. An Even Faster and More Unifying Algorithm for Comparing Trees via Unbalanced Bipartite Matchings. *Journal of Algorithms*, 40(2):212–233, 2001.

[10] Maddison, D.R., and K.-S. Schulz (eds.). The Tree of Life Web Project. `http://tolweb.org`, 1996-2006.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 373-384
www.stacs-conf.org

COMPLEXITY OF SOLUTIONS OF EQUATIONS OVER SETS OF NATURAL NUMBERS

ARTUR JEŻ [1] AND ALEXANDER OKHOTIN [2]

[1] Institute of Computer Science, University of Wrocław, Poland
E-mail address: aje@ii.uni.wroc.pl

[2] Department of Mathematics, University of Turku, Finland; Academy of Finland
E-mail address: alexander.okhotin@utu.fi

Abstract. Systems of equations over sets of natural numbers (or, equivalently, language equations over a one-letter alphabet) of the form $X_i = \varphi_i(X_1, \ldots, X_n)$ $(1 \leqslant i \leqslant n)$ are considered. Expressions φ_i may contain the operations of union, intersection and pairwise sum $A+B = \{x+y \mid x \in A, y \in B\}$. A system with an EXPTIME-complete least solution is constructed, and it is established that least solutions of all such systems are in EXPTIME. The general membership problem for these equations is proved to be EXPTIME-complete.

1. Introduction

The study of expressions over sets of numbers and of the computational complexity of their properties began in the paper by Stockmeyer and Meyer [17], who considered subsets of $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ as formal languages over a one-letter alphabet. In this case, concatenation of languages turns into a pairwise addition of elements of sets: $X+Y = \{x+y | x \in X, y \in Y\}$. Stockmeyer and Meyer established that the membership problem for expressions with union, intersection and addition is NP-complete.

Some extensions of this result were obtained by Yang [18], who considered integer circuits (that is, expressions in which subexpressions may be shared) with one more operation of pairwise multiplication, and established similar complexity results. A systematic study of complexity of expressions and circuits with different sets of operations was carried out by McKenzie and Wagner [9, 10].

In this paper we consider *equations over sets of natural numbers*, which are a more general device than expressions and circuits, and study the computational complexity of their least solutions, as well as of their membership problem. These equations naturally correspond to *language equations* over a one-letter alphabet. Language equations have recently become an active area of research, see a recent survey by Kunc [8]. In particular, unexpected hardness results on language equations have been obtained by Kunc [7] and by

(A. Jeż) Supported by MNiSW grant number N206 024 31/3826, 2006–2008, and by a short visit grant from the European Science Foundation under project AutoMathA, reference number 1763.
(A. Okhotin) Supported by the Academy of Finland under grant 118540.

Okhotin [15, 16], and this connection gives another motivation for our study. Recent results by Jeż [5] on the expressive power of *conjunctive grammars* provide a technical foundation for our results.

We consider equations in the *resolved form*

$$\begin{cases} X_1 &= \varphi_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= \varphi_n(X_1, \ldots, X_n) \end{cases} \tag{*}$$

in which every variable X_i assumes value of a set of nonnegative integers. The right-hand side φ_i of each equation may contain the operations of union, intersection and $+$, as well as singleton constants. Every such system has a least solution with respect to componentwise inclusion, which can be obtained by fixpoint iteration. Our result, established in Section 3, is a construction of a system (*), such that testing the membership of numbers in its least solution is an EXPTIME-hard problem (with the numbers given in binary notation). The result is obtained by a new kind of arithmetization of an alternating linear-space Turing machine. It is also shown that for every system (*) the membership of numbers in its least solution can be tested in exponential time, which makes the constructed set the hardest.

Let us compare our result to the existing results on expressions and circuits on sets of numbers. Previous research was concerned with the complexity of the general membership problem, where it was sufficient to encode an instance of some hard problem for numbers in an expression or a circuit. In our case, the task is to construct a system that represents a class of problems, while instances of that problem are to be encoded as numbers.

As compared to the research on language equations, our present approach studies a similar problem of constructing a representation of a hard set (cf. Kunc [7], Okhotin [15, 14], Jeż [5]). However, while encoding a computation of a Turing machine as a string over $\{a, b\}$ is an ordinary task, in our case we have to encode similar objects as numbers, that is, as strings over a one-letter alphabet. These strings have no apparent structure, and hence the proposed arithmetization is quite unobvious.

This result allows us to establish the complexity of the general membership problem for equations with $\{\cup, \cap, +\}$, which is stated as follows: *"Given a system and a number $n \geqslant 0$ in binary notation, determine whether n is in the first component of the least solution of the system"*. For integer expressions and integer circuits with the operations $\{\cup, \cap, +\}$, it is known from Stockmeyer and Meyer [17] and from McKenzie and Wagner [9, 10] that a similar problem is PSPACE-complete. Another weaker model are equations with $\{\cup, +\}$, that is, without intersection, for which the corresponding problem is NP-complete due to the result of Huynh [4] on the commutative case of the context-free grammars. In our case of equations with $\{\cup, \cap, +\}$, the general membership problem is EXPTIME-complete, which is established in Section 4. An exponential algorithm for solving this problem is given by a parsing algorithm on conjunctive grammars [13].

2. Language equations and conjunctive grammars

While our results are on the complexity of equations in sets of numbers, our methods are derived from the domain of formal language theory, in particular, from some recent results on language equations.

In language equations, the unknowns are formal languages over an alphabet Σ. If $|\Sigma| = 1$, they coincide with equations over sets of numbers, while for larger alphabets

they constitute a more general notion. The main object of this study are equations of the resolved form (*), in which variables assume values of sets of non-negative integers, and the right-hand sides may contain the operations of union, intersection and addition of sets. These equations obviously correspond to *language equations* over a one-letter alphabet with the operations of union, intersection and concatenation, and the recent results on language equations of this kind provide a theoretical foundation, as well as a second motivation, for the present research.

The first type of language equations to be studied were equations of the same form (*) containing union and concatenation, but no intersection: Ginsburg and Rice [3] established that these equations provide a natural semantics for the context-free grammars. Equations with added intersection therefore constitute a generalization of the context-free grammars.

Definition 2.1 (Okhotin [12])**.** A conjunctive grammar is a quadruple $G = (\Sigma, N, P, S)$, in which Σ and N are disjoint finite non-empty sets of terminal and nonterminal symbols respectively; P is a finite set of grammar rules, each of the form

$$A \to \alpha_1 \& \ldots \& \alpha_n \quad (\text{where } A \in N,\ n \geqslant 1 \text{ and } \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*)$$

while $S \in N$ is a nonterminal designated as the start symbol.

The semantics of conjunctive grammars is defined by the least solution of the following system of language equations:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \in P} \bigcap_{i=1}^{m} \alpha_i \quad (\text{for all } A \in N) \tag{2.1}$$

The component corresponding to each $A \in N$ is then denoted by $L_G(A)$, and $L(G)$ is defined as $L_G(S)$.

The operations used in the right-hand sides of systems (2.1) are union, intersection and concatenation. Since they are monotone and continuous, a least solution always exists and can be obtained by fixpoint iteration as

$$\bigsqcup_{i \geqslant 0} \varphi^i(\varnothing, \ldots, \varnothing), \tag{2.2}$$

where φ is the right-hand side of (2.1) as a vector operator on $|N|$-tuples of languages, while $\sqcup$ denotes pairwise union of vectors of sets.

An equivalent definition of conjunctive grammars can be given using term rewriting [12], which generalizes Chomsky's word rewriting. The importance of these grammars lies with the fact that their expressive power is substantially greater than that of the context-free grammars, while the generated languages can still be parsed in time $O(n^3)$, and the practical context-free parsing algorithms, such as recursive descent and generalized LR, admit generalization to conjunctive grammars without an increase in their complexity.

The question of whether conjunctive grammars can generate any non-regular unary language has been an open problem for some years, until recently solved by Jeż [5], who constructed a grammar for the language $\{a^{4^n} \mid n \geqslant 0\}$. Let us reformulate this grammar as the following resolved system of four equations over sets of numbers:

Example 2.2 (Jeż [5])**.** The system

$$\begin{cases} X_1 &= \big((X_2 + X_2) \cap (X_1 + X_3)\big) \cup \{\mathtt{1}\} \\ X_2 &= \big((X_{12} + X_2) \cap (X_1 + X_1)\big) \cup \{\mathtt{2}\} \\ X_3 &= \big((X_{12} + X_{12}) \cap (X_1 + X_2)\big) \cup \{\mathtt{3}\} \\ X_{12} &= \big((X_3 + X_3) \cap (X_1 + X_2)\big) \end{cases}$$

has least solution $X_i = \{\ell \mid \text{base-4 notation of } \ell \text{ is } i\mathtt{0}\ldots\mathtt{0}\}$, for $i = 1, 2, 3, 12$.

Sets of this kind can be conveniently specified by regular expressions for the corresponding sets of base-k notations of numbers, which in this case are $\mathtt{10}^*$, $\mathtt{20}^*$, $\mathtt{30}^*$ and $\mathtt{120}^*$, respectively. In the following we shall omit some parentheses in the right-hand sides of equations, and assume the following default precedence of operations: addition has the highest precedence, followed by intersection, and then by union with the least precedence.

Using the same technique in a more elaborate construction, a general theorem on the expressive power of unary conjunctive grammars was established. It can be reformulated for equations over sets of numbers as follows:

Theorem 2.3 (Jeż [5])**.** *For every $k \geqslant 2$ and for every finite automaton M over the alphabet $\{0, \ldots, k-1\}$ there exists a system of resolved language equations over $\mathbb{N}_0$ using $\cup, \cap, +$, such that its least solution is*

$$(S_1, S_2, \ldots, S_n),$$

where $S_i \subseteq \mathbb{N}_0$ and $S_1 = \{\ell \mid k\text{-ary notation of } \ell \text{ is in } L(M)\}$.

Let us note in passing a recent paper by Jeż and Okhotin [6] establishing a generalization of this result to a larger family of automata recognizing positional notations.

Though representing sets of numbers with a regular positional notation using this type of formal grammars was an unexpected and strong result in terms of language theory, it has no implications on computational complexity, as all these sets are computationally easy. More general representation theorem of Jeż and Okhotin [6] also does not imply any better complexity results than P-completeness, which, as the present paper shows, is much below the actual complexity of these equations.

Therefore, a new method of constructing such equations is needed to understand their complexity. This step is made in the next section, which introduces an arithmetization technique based upon addition of sets of numbers.

3. Representing an EXPTIME-complete language

In this section it will be shown that languages defined by least solutions of resolved language equations using $+$, $\cup$ and $\cap$ can be EXPTIME-complete, and this is the hardest language in this family. Denote this family by $EQ(\cup, \cap, +)$.

Theorem 3.1. *The family $EQ(\cup, \cap, +)$ is contained in EXPTIME and contains an EXPTIME-complete language.*

The proof is by constructing such a system of equations. The given system encodes a computation of a linear-bounded alternating Turing machine (ATM) It is known that such machines can recognize some EXPTIME-complete languages [2].

In our case we shall consider ATMs operating on a circular tape and moving to the right at every step. Its tape originally contains the input word, and the squares containing

it constitute all space available to the machine. Obviously, such machines are as powerful as linear-bounded ATMs of the general form.

Formally, such a machine is defined as $M = (\Omega, \Gamma, Q_E, Q_A, \delta, q_0, q_{fin})$, where Ω is the input alphabet, $\Gamma = \{a_0, a_1, \ldots, a_{\max}\} \supset \Omega$ is the tape alphabet, Q_E and Q_A are disjoint sets of existential and universal states, respectively, $Q = Q_E \cup Q_A$ and $q_0, q_{fin} \in Q$. Given an input $w \in \Omega^+$, M starts in state q_0 with the head over the first symbol of w. The transition function is $\delta : Q \times \Gamma \to 2^{Q \times \Gamma}$, and the head is moved one symbol to the right at every step. Once the head moves beyond the right-most symbol, it is moved back over the first symbol of w, maintaining its current state; this implements a circular tape. For technical reasons, assume that $(q, a') \notin \delta(q, a)$ for all $q \in Q$ and $a, a' \in \Sigma$, (that is, the machine never stays in the same state), and that $\delta(q, a) \neq \varnothing$ for all $q \in Q_A$ and $a \in \Sigma$.

Our construction of a system of equations over sets of numbers simulating a computation is based upon representing instantaneous descriptions of the ATM as *numbers*. We shall think of these numbers as written in base-$(8+|Q|+\max(|Q|+7, |\Gamma|))$ positional notation, and the entire argument is based upon mapping the symbols used by the machine to digits, and then using addition to manipulate individual digits in the positional notation of numbers. It must be noted that this positional notation is only a tool for our understanding of the constructions, while the actual equations deals with numbers as they are.

Let $\Sigma = \{0, 1, \ldots, 7 + |Q| + \max(|Q| + 7, |\Gamma|)\}$ be the alphabet of digits, and define the mapping of symbols to digits, $\langle\cdot\rangle : Q \cup \Gamma \to \Sigma$, as follows:

$$\langle q_i \rangle = 7 + i \quad (\text{for } q_i \in Q)$$
$$\langle a_i \rangle = 7 + |Q| + i \quad (\text{for } a_i \in \Gamma)$$

Furthermore, let $\langle Q \rangle = \{\langle q \rangle \mid q \in Q\}$ and $\langle \Gamma \rangle = \{\langle a \rangle \mid a \in \Gamma\}$. Now the tape of the ATM containing symbols $a_{i_1} \ldots a_{i_n}$, with the head over the j-th symbol and the machine in state q, is represented as the following string of digits:

$$0\langle a_{i_1} \rangle \ldots 0\langle a_{i_{j-1}} \rangle \langle q \rangle \langle a_{i_j} \rangle 0 \langle a_{i_{j+1}} \rangle \ldots 0\langle a_{i_n} \rangle 0 \in \Sigma^*$$

For technical reasons, configurations in which the head has just moved over the last symbol but has not yet jumped to the first position are considered separately, and will be represented as strings of the form

$$0\langle a_{i_1} \rangle \ldots 0\langle a_{i_n} \rangle \langle q \rangle,$$

where q is the current state. Note that digits denoting letters are written only in even positions, while odd positions are reserved for the states of the Turing machine. The set of all strings of digits representing valid encodings of tapes is specified by the following regular expression over Σ:

$$\text{Tape} = (0\langle \Gamma \rangle)^* \langle Q \rangle (\langle \Gamma \rangle 0)^* \setminus \langle Q \rangle$$

The set Tape should be a considered as a formal language over Σ, which will be used later as a part of representations of some sets of numbers. Subsets of this set representing tapes with different states will be denoted as follows:

$$\text{Tape}_u = \{w \mid w \in \text{Tape},\ u \text{ is a substring of } w\}$$
$$\text{Tape}_u^{\ell} = \{w \mid w \in \text{Tape},\ u \text{ is a prefix of } w\}$$

Besides the contents of the tape, the encoding for Turing machine configurations uses a counter of rotations of the circular tape. This counter specifies the number of passes

through the tape the machine is still allowed to make before it must halt. It is represented in binary notation using digits $\{\mathtt{0},\mathtt{1}\}$, and the set of valid counter representations is

$$\text{Counter} = \mathtt{1}\{\mathtt{0},\mathtt{1}\}^*$$

Normally the counter uses only digits $\{\mathtt{0},\mathtt{1}\}$, but in order to implement the incrementation of the counter we shall use strings with one digit $\mathtt{2}$ representing zero with carry. The set of valid representations of counters with a carry is

$$\text{Counter}' = \mathtt{1}\{\mathtt{0},\mathtt{1}\}^*\mathtt{2}\{\mathtt{0},\mathtt{1}\}^* \cup \mathtt{2}\{\mathtt{0},\mathtt{1}\}^*$$

For every string $c_{k-1}\dots c_0 \in \text{Counter} \cup \text{Counter}'$, define its value as

$$\text{Value}(c_{k-1}\dots c_0) = \sum_{j=0}^{k-1} c_j \cdot 2^j.$$

Now define the mapping from configurations of the Turing machine to numbers. A configuration with the tape contents, head position and current state given by a string of digits $w \in \text{Tape}$, and with the counter value given by $x \in \text{Counter}$ is represented by a string of digits

$$x\mathtt{55}w,$$

where two marker digits $\mathtt{55}$ separate the values. This string of digits in base-$|\Sigma|$ positional notation specifies a certain number, which accordingly represents the configuration.

The key property of this encoding is that *every transition of the ATM reduces the numerical value of its configuration.* Indeed, if the head is moved to the right, then a digit $\langle q\rangle$ is replaced with $\mathtt{0}$ and all other modifications are done on less signigicant digits. If the head jumps from the end to the beginning, then the counter is decremented, and since the counter occupies more significant positions in the number than the tape, this transition decreases the value of the configuration as well. This monotonicity allows us to encode dependence of configurations on each other by using addition of nonnegative numbers only.

The construction of equations representing the computation of the ATM begins with some expressions that will be used in the right-hand sides of equations. These expressions contain some constant sets of numbers given as regular languages over the alphabet Σ. Every such language represents the set of all numbers with $|\Sigma|$-ary notation of the given form. According to Theorem 2.3, every such set can be represented by a separate system of equations using only singleton constants. All these subsystems are assumed to be included in the constructed system, and each of the regular expressions in the system can be formally regarded as a reference to one of the auxiliary variables.

Definitions of a few of these regular languages incorporate positional notations of numbers obtained by subtracting one number from another. For convenience, these values are given in the form $u \boxminus v$, with $u, v \in \Sigma^*$ being positional notations of two numbers (the former shall be greater or equal to the latter). One can write, e.g., $(u \boxminus v)\mathtt{0}^*$ for the set of all numbers with their $|\Sigma|$-ary notation beginning with fixed digits determined by the given difference, followed with any number of zeroes.

$$\mathrm{Step}(X) = \Big(\bigcup_{\substack{q\in Q_E \\ a\in\Gamma}} \bigcup_{(q',a')\in\delta(q,a)} \mathrm{Move}_{q',a',q,a}(X) \Big) \cup \Big(\bigcup_{\substack{q\in Q_A \\ a\in\Gamma}} \bigcap_{(q',a')\in\delta(q,a)} \mathrm{Move}_{q',a',q,a}(X) \Big)$$

$$\begin{aligned}\mathrm{Move}_{q,a,q',a'}(X) &= \big(X \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_{\langle a\rangle\langle q\rangle}\big) + \big(\langle q'\rangle\langle a'\rangle \mathtt{0} \boxminus \langle a\rangle\langle q\rangle\big)(\mathtt{00})^* \\ &\quad \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_{\langle q'\rangle\langle a'\rangle} \\ \mathrm{Jump}(X) &= \bigcup_q \Big[\big(X \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}^{\ell}_{\langle q\rangle}\big) + (\mathtt{1000} \boxminus \langle q\rangle)(\mathtt{00})^+ + \langle q\rangle\Big] \\ &\quad \cap (\mathrm{Counter} \cup \mathrm{Counter}')\mathtt{55}\,\mathrm{Tape}_{\langle q\rangle} \\ \mathrm{Carry}(Y) &= \Big[\Big(\big(Y \cap \{\mathtt{0},\mathtt{1}\}^*\mathtt{2}\{\mathtt{0},\mathtt{1}\}^*\,\mathtt{55}\,\mathrm{Tape}\big) + \mathtt{10}^* \cap \{\mathtt{0},\mathtt{1}\}^*\mathtt{3}\{\mathtt{0},\mathtt{1}\}^*\,\mathtt{55}\,\mathrm{Tape}\Big) \\ &\quad + \big(\mathtt{10} \boxminus \mathtt{3}\big)\mathtt{0}^*\Big] \cap \big(\{\mathtt{0},\mathtt{1}\}^+ \cup \{\mathtt{0},\mathtt{1}\}^*\mathtt{2}\{\mathtt{0},\mathtt{1}\}^*\big)\,\mathtt{55}\,\mathrm{Tape}\end{aligned}$$

In addition, define the set of final configurations of the machine:

$$\mathrm{Final} = \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_{\langle q_{fin}\rangle}$$

The construction uses two variables, X and Y. Either variable represents the set of proper configurations of the machine, starting from which the machine accepts. The variable X represents configurations belonging to the set $\mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}$, while Y represents configurations from $(\mathrm{Counter} \cup \mathrm{Counter}')\mathtt{55}\,\mathrm{Tape}$, in which the counter may contain one carry digit $\mathtt{2}$ that needs to be propagated to higher positions. The equations, using the above auxiliary functions, are as follows:

$$X = \mathrm{Final} \cup \mathrm{Step}(X) \cup \big(Y \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}\big) \tag{3.1}$$

$$Y = \mathrm{Jump}(X) \cup \mathrm{Carry}(Y) \tag{3.2}$$

In order to determine the least solution of this system, let us first establish some properties of the auxiliary functions.

The first quite elementary property is their distributivity over infinite union, which allows us to study these operations as operations on individual numbers, and then infer their action on sets of numbers.

Lemma 1 (Distributivity)**.** Each function $f \in \{\mathrm{Move}_{q,a,q',a'}, \mathrm{Jump}, \mathrm{Carry}\}$ is distributive over infinite union, in the sense that $f(S) = \bigcup_{n\in S} f(\{n\})$ for every $S \subseteq \mathbb{N}_0$.

This follows from the fact that each of these expressions consists of intersections with constant sets, sums with constant sets and unions. On the other hand, note that if an expression contains intersections or sums of multiple expressions involving X, then it is not necessarily distributive over infinite union; in particular, Step need not be distributive.

One of the main technical devices used in these functions is addition of a constant set of numbers with $|\Sigma|$-ary notation $u\mathtt{0}^*$ (that is, a set $\{\, m \cdot |\Sigma|^i \mid i \geqslant 0 \,\}$) with one, two or three non-zero digits in u. The following lemma establishes that this addition can never rewrite the double markers $\mathtt{55}$, that is, every sum in which these markers are altered does not represent a valid tape contents. This means that every such addition manipulates the counter and the tape separately, and the changes do not mix.

Lemma 2 (Marker preservation)**.** For every $x, x' \in \{\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{3}\}^* \setminus \mathtt{0}\Sigma^*$ and $w, w' \in \mathrm{Tape}$, if $x'\mathtt{55}w' \in x\mathtt{55}w + (\Sigma^3 \cup \Sigma^2 \cup \Sigma)\mathtt{0}^*$, then $|w| = |w'|$.

The next statement describes the operation of Carry: applied to a configuration with the counter having a single carry digit 2, Carry changes this digit to 0 and increments the next digit, making it 1 or 2. Note that all operations are in $|\Sigma|$-ary notation. The tape contents is not altered.

Lemma 3 (Carry propagation)**.** For every $x \in \text{Counter}'$ and for every $w \in \text{Tape}$, $\text{Carry}\big(\{x\mathtt{55}w\}\big) = \{x'\mathtt{55}w\}$, where $x' \in \text{Counter} \cup \text{Counter}'$ and $\text{Value}(x') = \text{Value}(x)$. If $x' \in \text{Counter}'$, then the position of 2 in x' is greater than the position of 2 in x.

According to Lemma 3, Carry moves the carry by one position higher. The next lemma shows that sufficiently many iterations of Carry always eliminate the carry digit: given a counter with the notation $x = \widetilde{x}\mathtt{01}^{k-1}\mathtt{2}$, Carry^k transforms it to $x = \widetilde{x}\mathtt{10}^{k-1}\mathtt{0}$.

Lemma 4 (Termination of carry propagation)**.** For every $x \in \text{Counter} \cup \text{Counter}'$ and $w \in \text{Tape}$ there exists $x' \in \text{Counter}$ and $k \geqslant 0$, such that $\text{Carry}^k(x\mathtt{55}w) = x'\mathtt{55}w$ and $\text{Value}(x) = \text{Value}(x')$.

The next lemma states the functionality of Jump, which can be described as follows. If Jump is applied to a configuration in which the head scans over the first symbol, then the result of the operation is the *previous* configuration, in which the head is at the right-most position beyond the end of the string, while the value of the counter x is greater by 1.

Lemma 5. Let $x = \widetilde{x}c \in \text{Counter}$ with $c \in \{\mathtt{0}, \mathtt{1}\}$ and $w = \langle q\rangle\widetilde{w}\mathtt{0} \in \text{Tape}$ with $q \in Q$, that is, w encodes a configuration with the head over the first symbol. Then $\text{Jump}(x\mathtt{55}w) = \{\widetilde{x}(c+1)\mathtt{550}\widetilde{w}\langle q\rangle\}$.

For any string $\alpha \in \Sigma^*$ of a different form, $\text{Jump}(\alpha) = \varnothing$.

It follows from Lemma 5 that Jump is a reversible function, that is, the previous configuration given by $\text{Jump}(x\mathtt{55}w)$ corresponds to $x\mathtt{55}w$ only. This is stated as follows:

Lemma 6. Let $x'\mathtt{55}w' \in \text{Jump}(x\mathtt{55}w)$. Then $w' = \mathtt{0}\widetilde{w}\langle q\rangle$ and $w = \langle q\rangle\widetilde{w}\mathtt{0}$ for some state q, and $\text{Value}(x') = \text{Value}(x) + 1$.

Let us now proceed with specifying the action of Move, which represents symbol manipulation, head movement and state change of a Turing machine according to the membership of states and symbols specified in δ. Generally, when $\text{Move}_{q,a,q',a'}$ is applied to a valid configuration, it computes the *preceding configuration* of the machine. This configuration is unique because of the restriction built in $\text{Move}_{q,a,q',a'}$ in its subscripts. The symbols and states used as the subscript restrict its applicability to the following case: in the current configuration the machine is in state q and the symbol to the left rewritten at the previous step is a, while in the previous configuration the machine was in state q' and scanned the symbol a'. For all other configurations and in all other cases, the function produces the empty set.

Lemma 7. Let $q, q' \in Q$ and $a, a' \in \Gamma$. Let $x \in \text{Counter}$ and $w = \widehat{w}\mathtt{0}\langle a\rangle\langle q\rangle\widetilde{w} \in \text{Tape}$ for some $\widehat{w} \in (\mathtt{0}\langle\Gamma\rangle)^*$ and $\widetilde{w} \in (\langle\Gamma\rangle\mathtt{0})^*$. Then $\text{Move}_{q,a,q',a'}(x\mathtt{55}w) = x\mathtt{55}\widehat{w}\langle q'\rangle\langle a'\rangle\mathtt{0}\widetilde{w}$.

For every string $\alpha \in \Sigma^*$ of a different form, $\text{Move}_{q,a,q',a'}(\alpha) = \varnothing$.

Similarly to Lemma 6, reversibility of $\text{Move}_{q,a,q',a'}$ directly follows from Lemma 7.

Lemma 8. Let $x\mathtt{55}w \in \text{Move}_{q',a',q,a}(x'\mathtt{55}w')$. Then $w = \widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}$ and $w' = \widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w}$ for some $\widehat{w} \in (\mathtt{0}\langle\Gamma\rangle)^*$ and $\widetilde{w} \in (\langle\Gamma\rangle\mathtt{0})^*$, and $x = x'$.

The flow control of the alternating Turing machine includes existential and universal nondeterminism in the corresponding states, and a single step is in fact a disjunction or conjunction of several transitions as specified in Move. This logic is transcribed in the expression $\mathrm{Step}(X)$, which computes the set of all *previous configurations*, from which machines in a universal state make all their transitions to configurations in X and machines in an existential state make at least one of their transitions to some configuration in X. This implements one step of the computation of the machine, backwards.

Lemma 9. Let $x \in \mathrm{Counter}$ and $w \in \mathrm{Tape}$, let $q \in Q$ be the state encoded in w. Then $x\mathtt{55}w \in \mathrm{Step}(X)$ if and only if

- the configuration w has the head *not* in the position beyond the right-most symbol, that is, $w = \widehat{w}\langle q\rangle\widetilde{w}\mathtt{0}$ for some $\widehat{w}, \widetilde{w} \in \Sigma^*$.
- if $q \in Q_E$, then for some string w' encoding next configuration of the ATM there holds $x\mathtt{55}w' \in X$.
- if $q \in Q_A$, then for every string w' encoding next configuration of the ATM there holds $x\mathtt{55}w' \in X$.

Having established the formal meaning of the auxiliary operations, let us return to the equations. The equation for X states that a configuration leads to acceptance if and only if it is accepting itself (Final), or one can directly proceed from it to a configuration leading to acceptance ($\mathrm{Step}(X)$), or that it is a configuration obtained in Y. The equation for Y specifies circular rotation of the tape by $\mathrm{Jump}(X)$ and implements iterated carry propagation as in Lemma 4 by a self-reference $\mathrm{Carry}(Y)$. Altogether, the least solution of these equations corresponds to the computation of the machine as follows:

Lemma 10. Let (L_X, L_Y) be the least solution of the equations (3.1)–(3.2).

- $\ominus$ Let $x \in \mathrm{Counter}$, $w \in \mathrm{Tape}$ and $x\mathtt{55}w \in L_X$. Then M accepts starting from the configuration represented by w.
- $\oplus$ Conversely, if M accepts starting from the configuration represented by $w \in \mathrm{Tape}$, and the longest path in the tree of the accepting computation has length ℓ, then for each $x \in \mathrm{Counter}$ with $\mathrm{Value}(x) \geqslant \ell$, there holds $x\mathtt{55}w \in L_X$.

It remains to observe that the number of steps of the machine is exponentially bounded, hence the acceptance of a word by the machine is represented by the following number in the least solution of the constructed system:

Main Lemma. ATM M accepts a string $a_1 \dots a_n \in \Omega^+$ if and only if

$$\mathtt{1}^{2+\log n+\log(|\Gamma|)n+\log(|Q|)}\mathtt{55}\langle q_0\rangle\langle a_1\rangle\mathtt{0}\langle a_1\rangle\mathtt{0}\dots\langle a_n\rangle\mathtt{0} \in L_X.$$

Proof of Theorem 3.1. The system of equations constructed above has an EXPTIME-complete least solution.

To see that the least solution of every system is in EXPTIME, it is sufficient to represent it as a conjunctive grammar over a unary alphabet. Then, given a number n, its membership in the least solution can be tested by supplying the string a^n to a known cubic-time parsing algorithm for conjunctive grammars [12]. Its time is cubic in n, hence exponential in the length of the binary notation of n. ∎

Having established a solution complexity theorem for equations over sets of numbers, let us discuss its implications on conjunctive grammars over a one-letter alphabet.

Every conjunctive language is in P [12], and some conjunctive languages over a multiple-letter alphabet are known to be P-complete [14]. The case of a unary alphabet is special, as it is known that no sparse language, in particular no unary language, can be P-complete unless DLOGSPACE = P [11, 1], that is, unless the notion of P-completeness is trivial. However, from Theorem 3.1 one can infer the following result slightly weaker than P-completeness:

Corollary 3.2. *There exists a EXPTIME-complete set of numbers $S \subseteq \mathbb{N}$, such that the language $L = \{a^n | n \in S\}$ of unary notations of numbers from S is generated by a conjunctive grammar.*

Note that for every unary language generated by a conjunctive grammar, the corresponding set of numbers is in EXPTIME. The set constructed in Corollary 3.2 can thus be regarded as the computationally hardest among unary conjunctive languages.

A simple consequence of Corollary 3.2 refers to the complexity of parsing for conjunctive grammars.

Corollary 3.3. *Unless* PSPACE = EXPTIME*, there is no logarithmic-space parsing algorithm for conjunctive languages over a unary alphabet.*

4. The membership problem

Consider the general membership problem for our equations, stated as follows: "Given a system $X_i = \varphi_i(X_1, \ldots, X_m)$ and a number n in binary notation, determine whether n is in the first component of the least solution of the given system". Its complexity is now easy to establish.

Theorem 4.1. *The membership problem for resolved systems of equations over sets of numbers with operations $\{\cup, \cap, +\}$ is EXPTIME-complete.*

Proof. Membership in EXPTIME. The algorithm begins with representing the given system as a conjunctive grammar over a unary alphabet, with a linearly bounded blow-up. The given number n is represented as a string a^n with an exponential blow-up. Then it is sufficient to apply the known polynomial-time algorithm for solving the membership problem for conjunctive grammars [13].

The EXPTIME-hardness of the general membership problem immediately follows from Theorem 3.1 by fixing the system of equations. ∎

Let us conclude by comparing the complexity of the membership problem for expressions, circuits and equations, as well as the families of sets representable by their solutions. All known results are given in Table 1.

The new complexity results for the equations over sets of numbers naturally fit into the framework of the existing research. On the other hand, the new results on the expressive power of equations come in a sharp contrast with the previous work: these equations can represent non-trivial sets of numbers, which are computationally as hard as the general membership problem for this class.

It remains an open question, what is the exact family of sets of natural numbers defined by these equations. For instance, is it possible to represent the set of all primes?

	Representable sets	Membership problem
expressions with $\{\cup, +\}$	Finite	NP-complete [17]
circuits with $\{\cup, +\}$	Finite	NP-complete [4, 9, 10]
equations with $\{\cup, +\}$	Ultimately periodic	NP-complete [4]
expressions with $\{\cup, \cap, +\}$	Finite	PSPACE-complete [17]
circuits with $\{\cup, \cap, +\}$	Finite	PSPACE-complete [9, 10]
equations with $\{\cup, \cap, +\}$	$\subsetneq$ ***EXPTIME***, contains ***EXPTIME-complete*** *set*	***EXPTIME-complete***

Table 1: Comparison of formalisms over sets of integers.

References

[1] J.-Y. Cai, D. Sivakumar, "Sparse hard sets for P: resolution of a conjecture of Hartmanis". *Journal of Computer and System Sciences*, 58:2 (1999), 280–296.

[2] A. K. Chandra, D. C. Kozen, L. J. Stockmeyer, "Alternation", *Journal of the ACM*, 28:1 (1982) 114–133.

[3] S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.

[4] D. T. Huynh, "Commutative grammars: the complexity of uniform word problems", *Information and Control*, 57:1 (1983), 21–39.

[5] A. Jeż, "Conjunctive grammars can generate non-regular unary languages", *DLT 2007* (Turku, Finland, July 3–6, 2007), LNCS 4588, 242–253.

[6] A. Jeż, A. Okhotin, "Conjunctive grammars over a unary alphabet: undecidability and unbounded growth", *Computer Science in Russia* (CSR 2007, Ekaterinburg, Russia, September 3–7, 2007), LNCS 4649, 168–181.

[7] M. Kunc, "The power of commuting with finite sets of words", *Theory of Computing Systems*, 40:4 (2007), 521–551.

[8] M. Kunc, "What do we know about language equations?", *Developments in Language Theory* (DLT 2007, Turku, Finland, July 3–6, 2007), LNCS 4588, 23–27.

[9] P. McKenzie, K. Wagner, "The complexity of membership problems for circuits over sets of natural numbers", *20th Annual Symposium on Theoretical Aspects of Computer Science* (STACS 2003, Berlin, Germany, February 27–March 1, 2003), LNCS 2607, 571–582.

[10] P. McKenzie, K. Wagner, "The complexity of membership problems for circuits over sets of natural numbers", *Computational Complexity*, 16 (2007), to appear.

[11] M. Ogihara, "Sparse hard sets for P yield space-efficient algorithms", *Chicago J. Theor. Comput. Sci.*, 1996.

[12] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.

[13] A. Okhotin, "A recognition and parsing algorithm for arbitrary conjunctive grammars", *Theoretical Computer Science*, 302 (2003), 365–399.

[14] A. Okhotin, "The hardest linear conjunctive language", *Information Processing Letters*, 86:5 (2003), 247–253.

[15] A. Okhotin, "Decision problems for language equations with Boolean operations", *Automata, Languages and Programming* (ICALP 2003, Eindhoven, The Netherlands, June 30–July 4, 2003), LNCS 2719, 239–251.

[16] A. Okhotin, "Unresolved systems of language equations: expressive power and decision problems", *Theoretical Computer Science*, 349:3 (2005), 283–308.

[17] L. J. Stockmeyer, A. R. Meyer, "Word problems requiring exponential time", *STOC 1973*, 1–9.

[18] K. Yang, "Integer circuit evaluation is PSPACE-complete", *Computational Complexity 2000*, 204–211.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 385-396
www.stacs-conf.org

CARDINALITY AND COUNTING QUANTIFIERS ON OMEGA-AUTOMATIC STRUCTURES

ŁUKASZ KAISER [1], SASHA RUBIN [2], AND VINCE BÁRÁNY [1]

[1] Mathematische Grundlagen der Informatik, RWTH Aachen
E-mail address: {kaiser,vbarany}@informatik.rwth-aachen.de

[2] Department of Computer Science, University of Auckland
E-mail address: rubin@cs.auckland.ac.nz

Abstract. We investigate structures that can be represented by omega-automata, so called omega-automatic structures, and prove that relations defined over such structures in first-order logic expanded by the first-order quantifiers 'there exist at most $\aleph_0$ many', 'there exist finitely many' and 'there exist k modulo m many' are omega-regular. The proof identifies certain algebraic properties of omega-semigroups.

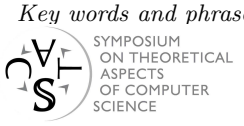
As a consequence an omega-regular equivalence relation of countable index has an omega-regular set of representatives. This implies Blumensath's conjecture that a countable structure with an ω-automatic presentation can be represented using automata on finite words. This also complements a very recent result of Hjörth, Khoussainov, Montalban and Nies showing that there is an omega-automatic structure which has no injective presentation.

1. Introduction

Automatic structures were introduced in [5] and later again in [6, 2] along the lines of the Büchi-Rabin equivalence of automata and monadic second-order logic. The idea is to encode elements of a structure $\mathfrak{A}$ via words or labelled trees (the codes need not be unique) and to represent the relations of $\mathfrak{A}$ via synchronised automata. This way we reduce the first-order theory of $\mathfrak{A}$ to the monadic second-order theory of one or two successors. In particular, the encoding of relations defined in $\mathfrak{A}$ by first order formulas are also regular, and automata for them can be computed from the original automata. Thus we have the fundamental fact that the first-order theory of an automatic structure is decidable.

Depending on the type of elements encoding the structure, the following natural classes of structures appear: automatic (finite words), ω-automatic (infinite words), tree-automatic (finite trees), and ω-tree automatic (infinite trees). Besides the obvious inclusions, for instance that automatic structures are also ω-automatic, there are still some some outstanding problems. For instance, a presentation over finite words or over finite trees can be transformed into one where each element has a unique representative.

Key words and phrases: ω-automatic presentations, ω-semigroups, ω-automata.



Kuske and Lohrey [9] point out an ω-regular equivalence relation (namely $\sim_{\mathrm{e}}$ stating that two infinite words are position-wise eventually equal) with no ω-regular set of representatives. Thus, unlike the finite-word case, injectivity can not generally be achieved by selecting a regular set of representatives from a given presentation. In fact, using topological methods it has recently been shown [4] that there are omega-automatic structures having no injective presentation. However, we are able to prove that every omega-regular equivalence relation having only countably many classes does allow to select an omega-regular set of unique representants. Therefore, every countable omega-automatic structure does have an injective presentation.

A related question raised by Blumensath [1] is whether every countable ω-automatic structure is also automatic. In Corollary 2.8 we confirm this by transforming the given presentation into an injective one, and then noting that an injective ω-automatic presentation of a countable structure can be "packed" into one over finite words.

All these results rest on our main contribution: a characterisation of when there exist countably many words x satisfying a given formula with parameters in a given ω-automatic structure $\mathfrak{A}$ (with no restriction on the cardinality of the domain of $\mathfrak{A}$ or the injectivity of the presentation). The characterisation is first-order expressible in an ω-automatic presentation of an extension of $\mathfrak{A}$ by $\sim_{\mathrm{e}}$. Hence we obtain an extension of the fundamental fact for ω-automatic structures to include cardinality and counting quantifiers such as 'there exists (un)countably many', 'there exists finitely many', and 'there exists k modulo m many'. This generalises results of Kuske and Lohrey [9] who achieve this for structures with *injective* ω-automatic presentations.

2. Preliminaries

By countable we mean finite or countably infinite. Let Σ be a finite alphabet. With Σ^* and Σ^ω we denote the set of finite, respectively ω-words over Σ. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, the empty word by ε, and for each $0 \leq i < |w|$ the ith symbol of w is written as $w[i]$. Similarly $w[n, m]$ is the factor $w[n]w[n+1]\cdots w[m]$ and $w[n, m)$ is defined by $w[n, m-1]$. Note that we start indexing with 0 and that for $u \in \Sigma^*$ we denote by u^n the concatenation of n number of us, in particular $u^\omega \in \Sigma^\omega$.

We consider relations on finite and ω-words recognised by multi-tape finite automata operating in a synchronised letter-to-letter fashion. Formally, *ω-regular relations* are those accepted by some finite non-deterministic automaton $\mathcal{A}$ with Büchi, parity or Muller acceptance conditions, collectively known as ω-automata, and having transitions labelled by m-tuples of symbols of Σ. Equivalently, $\mathcal{A}$ is a usual one-tape ω-automaton over the alphabet Σ^m accepting the *convolution* $\otimes\vec{w}$ of ω-words $w_1, \ldots, w_m$ defined by $\otimes\vec{w}[i] = (w_1[i], \ldots, w_m[i])$ for all i.

Words $u, v \in \Sigma^\omega$ have *equal ends*, written $u \sim_{\mathrm{e}} v$, if for almost all $n \in \mathbb{N}$, $u[n] = v[n]$. This is an important ω-regular equivalence relation. We overload notation so that for $S, T \subset \mathbb{N}$ we write $S \sim_{\mathrm{e}} T$ to mean for almost all $n \in \mathbb{N}$, $n \in S \iff n \in T$.

Example 2.1. The non-deterministic Büchi automaton depicted in Fig. 1 accepts the equal-ends relation on alphabet $\{0, 1\}$.

In the case of finite words one needs to introduce a padding end-of-word symbol $\Box \notin \Sigma$ to formally define convolution of words of different length. For simplicity, we shall identify each finite word $w \in \Sigma^*$ with its infinite padding $w^\Box = w\Box^\omega \in \Sigma_\Box^\omega$ where $\Sigma_\Box = \Sigma \cup \{\Box\}$.

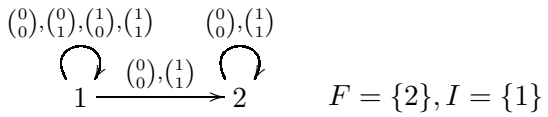


Figure 1: An automaton for the equal ends relation $\sim_{\mathrm{e}}$.

To avoid repeating the definition of automata for finite words, we say that a m-ary relation $R \subseteq (\Sigma^*)^m$ is *regular* (*synchronised rational*) whenever it is ω-regular over $\Sigma_\Box$.

2.1. Automatic structures

We now define what it means for a relational structure (we implicitly replace any structure with its relational counterpart) to have an (ω-)automatic presentation.

Definition 2.2 ((ω-)Automatic presentations)**.**
Consider a relational structure $\mathfrak{A} = (A, \{R_i\}_i)$ with universe $\mathsf{dom}(\mathfrak{A}) = A$ and relations R_i. A tuple of ω-automata $\mathfrak{d} = (\mathcal{A}, \mathcal{A}_\approx, \{\mathcal{A}_i\}_i)$ together with a surjective naming function $f : L(\mathcal{A}) \to A$ constitutes an *(ω-)automatic presentation* of $\mathcal{A}$ if the following criteria are met:

(i) the equivalence, denoted $\approx$, and defined by $\{(u, w) \in L(\mathcal{A})^2 \mid f(u) = f(w)\}$ is recognised by $\mathcal{A}_\approx$,
(ii) every $L(\mathcal{A}_i)$ has the same arity as R_i,
(iii) f is an isomorphism between $\mathfrak{A}_\mathfrak{d} = (L(\mathcal{A}), \{L(\mathcal{A}_i)\}_i)/_\approx$ and $\mathfrak{A}$.

The presentation is said to be *injective* whenever f is, in which case $\mathcal{A}_\approx$ can be omitted.

The relation $\approx$ needs to be a congruence of the structure $(L(\mathcal{A}), \{L(\mathcal{A}_i)\}_i)$ for item (iii) to make sense. In case $L(\mathcal{A})$ only consists of words of the form $w^\Box$ where $w \in \Sigma^*$, we say that the presentation is *automatic*. Call a structure ($\omega-$)*automatic* if it has an (ω-)automatic presentation.

The advantage of having an (ω-)automatic presentation of a structure lies in the fact that first-order (FO) formulas can be effectively evaluated using classical automata constructions. This is expressed by the following fundamental theorem.

Theorem 2.3. *(Cf.* [5], [6], [3].*)*

(i) There is an effective procedure that given an ($\omega-$)automatic presentation $\mathfrak{d}, f$ of a structure $\mathfrak{A}$, and given a FO-formula $\varphi(\vec{a}, \vec{x})$ with parameters $\vec{a}$ from $\mathfrak{A}$ (defining a k-ary relation R over $\mathfrak{A}$), constructs a k-tape synchronous ($\omega-$)automaton recognising $f^{-1}(R)$.
(ii) The FO-theory of every ($\omega-$)automatic structure is decidable.
(iii) The class of ($\omega-$)automatic structures is closed under FO-interpretations

Let FOC denote the extension of first-order logic with all quantifiers of the form

- $\exists^{(r \bmod m)} x \,.\, \varphi$ meaning that the number of x satisfying φ is finite and is congruent to r mod m;
- $\exists^\infty x \,.\, \varphi$ meaning that there are infinitely many x satisfying φ;
- $\exists^{\leq \aleph_0} x \,.\, \varphi$ and $\exists^{> \aleph_0} x \,.\, \varphi$ meaning that the cardinality of the set of all x satisfying φ is countable, or uncountable, respectively.

It has been observed that for *injective* (ω-)automatic presentations Theorem 2.3 can be extended from FO to FOC [8, 9]. Moreover, Kuske and Lohrey show that the cardinality of any set definable in FOC is either countable or equal to that of the continuum. Our main contribution is the following generalisation of their result.

Theorem 2.4. *The statements of Theorem 2.3 hold true for* FOC *over all (not necessarily injective) ω-automatic presentations.*

It is easily seen that finite-word automatic presentations can be assumed to be injective. This is achieved by restricting the domain of the presentation to a regular set of representatives of the equivalence involved. This can be done effectively, e.g. by selecting the length-lexicographically least word of every class.

This brings us to the question which ω-automatic structures allow an injective ω-automatic presentation. In [9] Kuske and Lohrey have pointed out that not every ω-regular equivalence has an ω-regular set of representatives. In particular, the following Lemma shows that the *equal-ends relation* $\sim_e$ of Example 2.1 is a counterexample.

Lemma 2.5 ([9, Lemma 2.4])**.** *Let $\mathcal{A}$ be a Büchi automaton with n states over $\Sigma \times \Gamma$ and let $u \in \Sigma^\omega$ be given. Consider the set $V = \{v \in \Gamma^\omega \mid u \otimes v \in L(\mathcal{A})\}$. Then V is uncountable if and only if $|V/\sim_e| > n$, otherwise it is finite or countable.*

The lemma implies that an ω-regular set is countable if and only if it meets only finitely many equal-ends-classes. In this case each of its members is ultimately periodic with one of finitely many periods.

Corollary 2.6. *An ω-regular set is countable iff it can be written as a finite union of sets of the form $U_j \cdot (w_j)^\omega$ with each U_j a regular set of finite words and each w_j a finite word.*

A related question raised by Blumensath [1] is whether every *countable* ω-automatic structure is also automatic. It is easy to see that every *injective* ω-automatic presentation of a countable structure can be "packed" into an automatic presentation.

Proposition 2.7. *(*[1, Theorem 5.32]*) Let $\mathfrak{d}$ be an injective ω-automatic presentation of a countable structure $\mathcal{A}$. Then, an (injective) automatic presentation $\mathfrak{d}'$ of $\mathcal{A}$ can be effectively constructed.*

In our proof of Theorem 2.4 we identify a property of finite semigroups that recognise transitive relations (Lemma 3.3 item (3)) that allows us to drop the assumption of injectivity in the previous statement. We are thus able to answer the question of Blumensath.

Corollary 2.8. *A countable structure is ω-automatic if and only if it is automatic. Transforming a presentation of one type into the other can be done effectively.*

2.2. ω-Semigroups

The fundamental correspondence between recognisability by finite automata and by finite semigroups has been extended to ω-regular sets. This is based on the notion of *ω-semigroups*. Rudimentary facts on ω-semigroups are well presented in [10]. We only mention what is most necessary.

An ω-semigroup $S = (S_f, S_\omega, \cdot, *, \pi)$ is a two-sorted algebra, where $(S_f, \cdot)$ is a semigroup, $* : S_f \times S_\omega \mapsto S_\omega$ is the *mixed product* satisfying for every $s, t \in S_f$ and every $\alpha \in S_\omega$ the equality

$$s \cdot (t * \alpha) = (s \cdot t) * \alpha$$

and where $\pi : S_f^\omega \mapsto S_\omega$ is the *infinite product* satisfying

$$s_0 \cdot \pi(s_1, s_2, \ldots) = \pi(s_0, s_1, s_2, \ldots)$$

as well as the associativity rule

$$\pi(s_0, s_1, s_2, \ldots) = \pi(s_0 s_1 \cdots s_{k_1}, s_{k_1+1} s_{k_1+2} \cdots s_{k_2}, \ldots)$$

for every sequence $(s_i)_{i\geq 0}$ of elements of S_f and every strictly increasing sequence $(k_i)_{i\geq 0}$ of indices. For $s \in S_f$ we denote $s^\omega = \pi(s, s, \ldots)$.

Morphisms of ω-semigroups are defined to preserve all three products as expected. There is a natural way to extend finite semigroups and their morphisms to ω-semigroups. As in semigroup theory, idempotents play a central role in this extension. An *idempotent* is a semigroup element $e \in S$ satisfying $ee = e$. For every element s in a finite semigroup the sub-semigroup generated by s contains a unique idempotent s^k. The least $k > 0$ such that s^k is idempotent for every $s \in S_f$ is called the *exponent* of the semigroup S_f and is denoted by π. Another useful notion is absorption of semigroup elements: say that s *absorbs* t *(on the right)* if $st = s$.

There is also a natural extension of the free semigroup Σ^+ to the ω-semigroup $(\Sigma^+, \Sigma^\omega)$ with $*$ and π determined by concatenation. An ω-semigroup $S = (S_f, S_\omega)$ *recognises* a language $L \subseteq \Sigma^\omega$ via a morphism $\phi : (\Sigma^+, \Sigma^\omega) \to (S_f, S_\omega)$ if $\phi^{-1}(\phi(L)) = L$. This notion of recognisability coincides, as for finite words, with that by non-deterministic Büchi automata. In [10] constructions from Büchi automata to ω-semigroups and back are also presented.

Theorem 2.9 ([10])**.**
A language $L \subseteq \Sigma^\omega$ is ω-regular iff it is recognised by a finite ω-semigroup.

We note that this correspondence allows one to engage in an algebraic study of varieties of ω-regular languages, and also has the advantage of hiding complications of cutting apart and stitching together runs of Büchi automata as we shall do. This is precisely the reason that we use this algebraic framework. Most remarkably, one does not need to understand the exact relationship between automata and ω-semigroups and the technical details of the constructions behind Theorem 2.9 to comprehend our proof. An alternative approach, though likely less advantageous, would be to use the composition method, which is closer in spirit to ω-semigroups than to automata. [1]

3. Cardinality and modulo counting quantifiers

This section is devoted to establishing the key to Theorem 2.4 announced earlier.

We characterise when there exist countably many words x satisfying a given formula *with parameters* $\varphi(x, \vec{z})$ in some ω-automatic structure $\mathfrak{A}$. The characterisation is first-order expressible in an ω-automatic extension of $\mathfrak{A}$ by the equal-ends relation $\sim_{\mathrm{e}}$.

So, fix an ω-automatic presentation of some $\mathfrak{A}$ with congruence $\approx$, and a first-order formula $\varphi(x, \vec{z})$ in the language of $\mathfrak{A}$ with x and $\vec{z}$ free variables.

Proposition 3.1. *There is a constant C, computable from the presentation $\mathfrak{d}$, so that for all tuples $\vec{z}$ of infinite words the following are equivalent:*

(1) $\varphi(-, \vec{z})$ is satisfiable and $\approx$ restricted to the domain $\varphi(-, \vec{z})$ has countably many equivalence classes.

[1] Define T_f resp. T_ω as the sets of bounded (in terms of quantifier rank) theories of finite, respectively, of ω-words. The composition theorem ensures that $\cdot$, $*$, π can naturally be defined on bounded theories.

(2) *there exist C-many words $x_1, \cdots, x_C$ each satisfying $\varphi(-, \vec{z})$, so that every x satisfying $\varphi(-, \vec{z})$ is $\approx$-equivalent to some $y \sim_e x_i$. Formally, the structure $(\mathfrak{A}, \approx, \sim_e)$ models the sentence below.*

$$\forall \vec{z} \left(\exists^{\leq \aleph_0} w \,.\, \varphi(w, \vec{z}) \longleftrightarrow \exists x_1 \dots x_C \left(\bigwedge_i \varphi(x_i, \vec{z}) \wedge \forall x \varphi(x, \vec{z}) \to \exists y (x \approx y \wedge \bigvee_i y \sim_e x_i) \right) \right)$$

Proof. Suppose $\mathfrak{d}$, $\mathfrak{A}$, and φ are given. Define C to be c^2, where c is the size of the largest ω-semigroup corresponding to any of the given automata (from the presentation or corresponding to φ). Now fix parameters $\vec{z}$. From now on, $\approx$ denotes the equivalence relation $\approx$ restricted to domain $\varphi(-, \vec{z})$.

$2 \to 1$: Condition 2 and the fact that every $\sim_{\mathrm{e}}$-class is countable imply that all words satisfying $\varphi(-, \vec{z})$ are contained in a countable number of $\approx$-classes.

$1 \to 2$: We prove the contra-positive in three steps.

If $\varphi(-, \vec{z})$ is satisfiable then the negation of condition 2 implies that there are $C + 1$ many words $x_0, \dots, x_C$ each satisfying $\varphi(-, \vec{z})$, and so that for $i, j \leq C$, $i \neq j$, the $\approx$-class of x_j does not meet the $\sim_{\mathrm{e}}$-class of x_i. In particular, the x_is are pairwise $\not\sim_{\mathrm{e}}$.

The plan is to produce uncountably many pairwise non-$\approx$ words that satisfy $\varphi(-, \vec{z})$. In the first 'Ramsey step', similar to what is done in [9], we find two words from the given C many, say $x_1, x_2 \in \Sigma^*$, and a factorisation $H \subset \mathbb{N}$ so that both words behave the same way along the factored sub-words with respect to the $\approx$- and φ-semigroups. In the second 'Coarsening step' we identify a technical property of finite semigroups recognising transitive relations. This allows us to produce an altered factorisation G and new, well-behaving words y_1, y_2. In the final step, the new words are 'shuffled along G' to produce continuum many pairwise non-$\approx$ words, each satisfying $\varphi(-, \vec{z})$.

3.1. Ramsey step

This step effectively allows us to discard the parameters $\vec{z}$. Before we use Ramsey's theorem, we introduce a convenient notation to talk about factorisations of words.

Definition 3.2. Let $A = a_1 < a_2 < \cdots$ be any subset of $\mathbb{N}$ and $h : \Sigma^* \to S$ be a morphism into a finite semigroup S. For an ω-word $\alpha \in \Sigma^\omega$, and element $e \in S$, say that A *is an h, e-homogeneous factorisation of* α if for all $n \in \mathbb{N}^+$, $h\big(\alpha[a_n, a_{n+1})\big) = e$.

Observe that

(1) if A is an h, s-homogeneous factorisation of α and $k \in \mathbb{N}^+$ then the set $\{a_{ki}\}_{i \in \mathbb{N}^+}$ is an h, s^k-homogeneous factorisation of α.
(2) if A is an h, e-homogeneous factorisation of α and e is idempotent, then every infinite $B \subset A$ is also an h, e-homogeneous factorisation of α.

In the following we write w^φ and $w^\approx$ to denote the image of w under the semigroup morphism into the finite semigroup associated to φ and $\approx$, respectively, as determined by the presentation. Accordingly, we will speak of e.g. φ, s_i-homogeneous factorisations.

Let us now colour every $\{n, m\} \in [\mathbb{N}]^2$, say $n < m$, by the tuple of ω-semigroup elements

$$\langle \ \big(\otimes (x_i, \vec{z})[n, m)^\varphi \big)_{0 \leq i \leq C} \ , \ \big(\otimes (x_i, x_j)[n, m)^\approx \big)_{0 \leq i \leq j \leq C} \ \rangle.$$

By Ramsey's theorem there exists infinite $H \subset \mathbb{N}$ and a tuple of ω-semigroup elements

$$\big\langle (s_i)_{1 \leq i \leq C}, (t_{(i,j)})_{1 \leq i \leq j \leq C} \big\rangle$$

so that for all $0 \leq i \leq j \leq C$,

- H is a φ, s_i-homogeneous factorisation of the word $\otimes(x_i, \vec{z})$,
- H is $\approx, t_{(i,j)}$-homogeneous factorisation of the word $\otimes(x_i, x_j)$.

Note that by virtue of additivity of our colouring and Ramsey's theorem each of the s_i and $t_{(i,j)}$ above are idempotents. Note that since there are at most c-many s_is and c-many $t_{(i,i)}$s there are at most c^2 many pairs $(s_i, t_{(i,i)})$ and so there must be two indices, we may suppose 1 and 2, with $s_1 = s_2$ and $t_{(1,1)} = t_{(2,2)}$.

3.2. Coarsening step

For technical reasons we now refine H and alter x_1, x_2 so that the semigroup elements have certain additional properties.

To start with, using the fact that $x_1 \not\sim_{\mathrm{e}} x_2$ and our observation on coarsenings, we assume without loss of generality that H is coarse enough so that $x_1[h_n, h_{n+1}) \neq x_2[h_n, h_{n+1})$ for all $n \in \mathbb{N}$.

Lemma 3.3. *There exists a subset $G \subset H$, listed as $g_1 < g_2 < \cdots$, and ω-words y_1, y_2 with the following properties:*

(1) The words y_1 and y_2 are neither $\approx$-equivalent nor $\sim_e$-equivalent, and each satisfies $\varphi(-, \vec{z})$.

(2) There exists an idempotent φ-semigroup element s such that G is a φ, s-homogeneous factorisation for each of $\otimes(y_1, \vec{z})$ and $\otimes(y_2, \vec{z})$.

(3) There exist idempotent $\approx$-semigroup elements $t, t^{\uparrow}, t^{\downarrow}$ so that for $y_j \in \{y_1, y_2\}$

- *both $t^{\uparrow}$ and $t^{\downarrow}$ absorb t*
- *$\otimes(y_j, y_j)[0, g_1)^{\approx}$ absorbs t*
- *G is an $\approx, t$-homogeneous factorisation of $\otimes(y_j, y_j)$*
- *G is an $\approx, t^{\uparrow}$-homogeneous factorisation of $\otimes(y_1, y_2)$*
- *G is an $\approx, t^{\downarrow}$-homogeneous factorisation of $\otimes(y_2, y_1)$.*

Proof. Define ω-words $y_1 := x_2[0, h_2)x_1[h_2, \infty)$, and y_2 by

$$\begin{aligned} y_2[0, h_2) &:= x_2[0, h_2) \text{ and} \\ y_2[h_{2n}, h_{2n+2}) &:= x_2[h_{2n}, h_{2n+1})x_1[h_{2n+1}, h_{2n+2}) \text{ for } n > 0. \end{aligned}$$

Item 1. Clearly, $y_1 \not\sim_{\mathrm{e}} y_2$ and each $y_j \in \{y_1, y_2\}$ satisfies $\varphi(y_j, \vec{z})$ since by homogeneity and $s_1 = s_2$

$$\begin{aligned} \otimes(y_1, \vec{z})^{\varphi} &= \otimes(x_2, \vec{z})[0, h_2)^{\varphi} s_1^{\omega} \\ &= \otimes(x_2, \vec{z})[0, h_2)^{\varphi} s_2^{\omega} \\ &= \otimes(x_2, \vec{z})^{\varphi} \end{aligned}$$

and similarly

$$\begin{aligned} \otimes(y_2, \vec{z})^{\varphi} &= \otimes(x_2, \vec{z})[0, h_2)^{\varphi} (s_2 s_1)^{\omega} \\ &= \otimes(x_2, \vec{z})[0, h_2)^{\varphi} s_2^{\omega} \\ &= \otimes(x_2, \vec{z})^{\varphi} \end{aligned}$$

Next we check that $y_1 \not\approx y_2$.

$$\begin{aligned}
\otimes(y_1,y_2)^{\approx} &= \pi_{\approx}\big(\otimes(x_2,x_2)[0,h_2)^{\approx},\ \big(\otimes(x_1,x_2)[h_{2n},h_{2n+1})^{\approx},\ \otimes(x_1,x_1)[h_{2n+1},h_{2n+2})^{\approx}\big)_{n\in\mathbb{N}^+}\big)\\
&= \otimes(x_2,x_2)[0,h_1)^{\approx}\, t_{(2,2)}\ (t_{(1,2)}t_{(1,1)})^{\omega}\\
&= \otimes(x_2,x_2)[0,h_1)^{\approx}\, t_{(2,2)}t_{(2,2)}\ (t_{(1,2)}t_{(1,1)})^{\omega}\\
&= \otimes(x_2,x_2)[0,h_1)^{\approx}\, t_{(2,2)}t_{(2,2)}\ (t_{(1,2)}t_{(2,2)})^{\omega}\\
&= \otimes(x_2,x_2)[0,h_1)^{\approx}\, t_{(2,2)}\ (t_{(2,2)}t_{(1,2)})^{\omega}\\
&= \pi_{\approx}\big(\otimes(x_2,x_2)[0,h_2)^{\approx},\ \big(\otimes(x_2,x_2)[h_{2n},h_{2n+1})^{\approx},\ \otimes(x_1,x_2)[h_{2n+1},h_{2n+2})^{\approx}\big)_{n\in\mathbb{N}^+}\big)\\
&= \otimes(y_2,x_2)^{\approx}
\end{aligned}$$

Thus, if $y_1 \approx y_2$ then also $y_2 \approx x_2$ and so **by transitivity** $y_1 \approx x_2$. But since $y_1 \sim_{\mathrm{e}} x_1$, the $\approx$-class of x_2 meets the $\sim_{\mathrm{e}}$-class of x_1, contradicting the initial choice of the x_is.

Items 2 and 3. Define intermediate semigroup elements $q := s_1$, $r := t_{(1,1)}$, $r^{\uparrow} := t_{(1,2)}t_{(1,1)}$ and $r^{\downarrow} := t_{(2,1)}t_{(1,1)}$. Then

(1) both $r^{\uparrow}$ and $r^{\downarrow}$ absorb r, since $t_{(1,1)}$ is idempotent;
(2) $\otimes(y_j,y_j)[0,h_2)^{\approx} = \otimes(y_j,y_j)[0,h_1)^{\approx}t_{(2,2)}$ and thus absorbs r (for $y_j \in \{y_1,y_2\}$).

In this notation, for all $i \in \mathbb{N}^+$ and $y_j \in \{y_1,y_2\}$,

- $\otimes(y_j,\vec{z})[h_{2i},h_{2i+2})^{\varphi}$ is $qq = q$,
- $\otimes(y_j,y_j)[h_{2i},h_{2i+2})^{\approx}$ is $rr = r$,
- $\otimes(y_1,y_2)[h_{2i},h_{2i+2})^{\approx}$ is $t_{(1,2)}t_{(1,1)} = r^{\uparrow}$,
- $\otimes(y_2,y_1)[h_{2i},h_{2i+2})^{\approx}$ is $t_{(2,1)}t_{(1,1)} = r^{\downarrow}$.

Finally, define the set $G := \{h_{2ki}\}_{i>1}$, i.e. $g_i = h_{2k(i+1)}$, and the semigroup elements $t := r^k$, $t^{\uparrow} := (r^{\uparrow})^k$, $t^{\downarrow} := (r^{\downarrow})^k$ and $s := q^k$. The extra multiple of k (defined as the product of the exponents of the give semigroups for $\sim_{\mathrm{e}}$ and $\approx$) ensures all these semigroup elements (in particular $t^{\uparrow}$ and $t^{\downarrow}$) are idempotent. We now verify the absorption properties:

$$t^{\uparrow}t = r^{\uparrow k}r^k = r^{\uparrow k} = t^{\uparrow} \qquad \text{because } r^{\uparrow} \text{ absorbs } r$$

Similarly, $t^{\downarrow}t$ absorbs t. Further, since $g_1 = h_{4k}$, we have

$$\begin{aligned}
\otimes(y_j,y_j)[0,g_1)^{\approx} &= \otimes(y_j,y_j)[0,h_2)^{\approx} \otimes (y_j,y_j)[h_2,h_{4k})^{\approx}\\
&= \otimes(y_j,y_j)[0,h_2)^{\approx} r^{4k-2}\\
&= \otimes(y_j,y_j)[0,h_2)^{\approx} r^{3k-2}t
\end{aligned}$$

and thus absorbs t.

Finally we verify the homogeneity properties: G is an $\approx, t^{\downarrow}$-homogeneous factorisation of $\otimes(y_2,y_1)$ since for $i \in \mathbb{N}^+$

$$\otimes(y_2,y_1)[g_i,g_{i+1})^{\approx} \;=\; \otimes(y_2,y_1)[h_{2k(i+1)},h_{2k(i+2)})^{\approx} \;=\; (r^{\downarrow})^k \;=\; t^{\downarrow}.$$

The other cases are similar. ∎

3.3. Shuffling step

We continue the proof of Proposition 3.1 by 'shuffling' the words y_1 and y_2 along G resulting in continuum many pairwise distinct words that are pairwise not $\approx$-equivalent, each satisfying $\varphi(-, \vec{z})$. To this end, define for $S \subset \mathbb{N}^+$ the 'characteristic word' χ_S by

$$\begin{aligned} \chi_S[0, g_1) &:= y_2[0, g_1) \text{ , and} \\ \chi_S[g_n, g_{n+1}) &:= \begin{cases} y_2[g_n, g_{n+1}) & \text{if } n \in S \\ y_1[g_n, g_{n+1}) & \text{otherwise} \end{cases} \end{aligned}$$

First note that $\mathfrak{A} \models \varphi(\chi_S, \vec{z})$. Indeed, by Lemma 3.3 item 2

$$\begin{aligned} \otimes(\chi_S, \vec{z})^\varphi &= \otimes(y_2, \vec{z})[0, g_1)^\varphi s^\omega \\ &= \otimes(y_2, \vec{z})^\varphi \end{aligned}$$

and $\mathfrak{A} \models \varphi(y_2, \vec{z})$ by Lemma 3.3 item 1. Moreover, for $S \not\sim_\mathrm{e} T$ the construction gives that $\chi_S \not\sim_\mathrm{e} \chi_T$. This is due our initial choice of $x_1 \not\sim_\mathrm{e} x_2$ and the assumption that the factorisation $(h_n)_n$ is coarse enough so that $x_1[h_n, h_{n+1}) \neq x_2[h_n, h_{n+1})$ and therefore also $y_1[g_n, g_{n+1}) \neq y_2[g_n, g_{n+1})$ for all n.

The following two lemmas establish that if $S \not\sim_\mathrm{e} T$ then $\chi_S \not\approx \chi_T$.

Write $x_{\circ\bullet}$ for the word $\chi_{2\mathbb{N}^+}$, and $x_{\bullet\circ}$ for $\chi_{2\mathbb{N}^+-1}$ and let p denote $\otimes(y_2, y_2)[0, g_1)^\approx$.

Lemma 3.4. *For all $S \not\sim_e T$,*

$$\otimes(\chi_S, \chi_T)^\approx = \begin{cases} \otimes(x_{\circ\bullet}, x_{\bullet\circ})^\approx & \text{or} \\ \otimes(x_{\bullet\circ}, x_{\circ\bullet})^\approx \end{cases}$$

Proof. Define semigroup-elements p_n for $n \in \mathbb{N}$ by

$$p_n := \begin{cases} t^\downarrow & \text{if } n \in S \setminus T \\ t^\uparrow & \text{if } n \in T \setminus S \\ t & \text{otherwise} \end{cases}$$

Let m be the smallest number in $S \triangle T$. Suppose that $m \in S \setminus T$. Because both $t^\uparrow$ and $t^\downarrow$ are idempotent and since t is absorbed by both p, $t^\uparrow$ and $t^\downarrow$ we have

$$\begin{aligned} \otimes(\chi_S, \chi_T)^\approx &= \pi_\approx(p, (p_n)_{n\in\mathbb{N}}) = p(t^\downarrow t^\uparrow)^\omega \\ &= \otimes(x_{\bullet\circ}, x_{\circ\bullet})^\approx \end{aligned}$$

and the case that $m \in T \setminus S$ similarly results in $\otimes(x_{\circ\bullet}, x_{\bullet\circ})^\approx$. ∎

Lemma 3.5. $x_{\circ\bullet} \not\approx x_{\bullet\circ}$.

Proof. Define an intermediate word $x_{\circ\bullet\circ\circ} := \chi_{4\mathbb{N}^+-2}$. By computations similar to the above we find that

$$\begin{aligned} \otimes(x_{\bullet\circ}, x_{\circ\bullet\circ\circ})^\approx &= p(t^\downarrow t^\uparrow t^\downarrow t)^\omega = p(t^\downarrow t^\uparrow t^\downarrow)^\omega = p(t^\downarrow t^\uparrow)^\omega \\ &= \otimes(x_{\bullet\circ}, x_{\circ\bullet})^\approx \end{aligned}$$

and

$$\begin{aligned} \otimes(x_{\circ\bullet}, x_{\circ\bullet\circ\circ})^\approx &= p(tttt^\downarrow)^\omega = p(t^\downarrow)^\omega \\ &= \otimes(y_2, y_1)^\approx \end{aligned}$$

Therefore, if $x_{\bullet\circ} \approx x_{\circ\bullet}$ then also $x_{\bullet\circ} \approx x_{\circ\bullet\circ\circ}$ and so **by symmetry** and **by transitivity** $x_{\circ\bullet} \approx x_{\circ\bullet\circ\circ}$. But in this case also $y_2 \approx y_1$, contradicting Lemma 3.3 item 1. ∎

There are continuum many classes in $\mathcal{P}(\mathbb{N})/\sim_e$, thus there is a continuum of pairwise not $\approx$-equivalent words χ_S each satisfying $\varphi(-, \vec{z})$. This completes the proof of Proposition 3.1. ∎

4. Consequences

Theorem 2.4 *The statements of Theorem 2.3 hold true for* FOC *over all (not necessarily injective) ω-automatic presentations.*

Proof. We prove item (i) from which the rest of the theorem follows immediately. We inductively eliminate occurrences of cardinality and modulo-counting quantifiers in the following way.

The countability quantifier $\exists^{\leq \aleph_0}$ and uncountability quantifier $\exists^{>\aleph_0}$ can be eliminated (in an extension of the presentation by $\sim_{\mathrm{e}}$) by the formula given in Proposition 3.1.

For the remaining quantifiers we further expand the presentation with the ω-regular relations

- $\pi(a, b, c)$ saying that $a \sim_{\mathrm{e}} b \sim_{\mathrm{e}} c$ and the last position where a differs from c is no larger than the last position where b differs from c, and
- $\lambda(a, b, c)$ saying that $\pi(a, b, c)$ and $\pi(b, a, c)$ and, writing k for this common position, the word $a[0, k]$ is lexicographically smaller than the word $b[0, k]$.

Now $\exists^{<\infty}. \varphi(x, \vec{z})$ is equivalent to

$$\exists x_1 \cdots x_C \, \Psi(x_1, \cdots, x_C, \vec{z})$$

where Ψ expresses that $x_1, \cdots x_C$ satisfy $\varphi(-, \vec{z})$ and there exists a position, say $k \in \mathbb{N}$, so that every $\approx$-class contains a word satisfying $\varphi(-, \vec{z})$ that coincides with one of the x_i from position k onwards. This additional condition can be expressed by

$$\exists y_1 \cdots y_C \forall x \exists y \left(\varphi(x, \vec{z}) \to x \approx y \wedge \bigvee_i \pi(y, y_i, x_i) \right)$$

Consequently, $\exists^{(r \bmod m)} x \, . \, \varphi(x, \vec{z})$ can be eliminated since we can pick out unique representatives of the $\approx$-classes as those x so that, writing $i(w)$ for the smallest index i for which $w \sim_{\mathrm{e}} x_i$, for every $y \neq x$ in the same $\approx$-class as x, either

- $i(x) < i(y)$, or
- $i(x) = i(y)$ and $\lambda(x, y, x_{i(x)})$.

Now we can apply the construction of [9] or [8] for elimination of the $\exists^{(r \bmod m)}$ quantifier. ∎

As a corollary of Proposition 3.1 we obtain that for every omega-regular equivalence with countably many classes a set of unique representants is definable.

Corollary 4.1. *Let $\approx$ be an ω-automatic equivalence relation on Σ^ω. There is a constant C, depending on the presentation, so that the following are equivalent:*

(1) $\approx$ has countably many equivalence classes.
(2) there exist C many $\sim_e$-classes so that every $\approx$-class has non-empty intersection with at least one of these C.

In this case there is an ω-regular set of representatives of $\approx$. Moreover an automaton for this set can be effectively found given an automaton for $\approx$.

Proof. The first two items are simply a specialisation of Proposition 3.1. We get the representatives as follows.

Write A for the domain of $\approx$ and consider the formula $\psi(x_1, \cdots, x_C)$ with free variables $x_1, \cdots, x_C$:

$$\bigwedge_i x_i \in A \wedge (\forall x \in A)(\exists y)\,[x \approx y \wedge \bigvee_i y \sim_{\mathrm{e}} x_i]$$

The relation defined by ψ is ω-regular since it is a first order formula over ω-regular relations. By assumption it is non-empty. Thus it contains an ultimately periodic word of the form $\otimes(a_1, \cdots, a_C)$. Thus each of these a_is is ultimately periodic; say $a_i = v_i(u_i)^\omega$.

Then every x has an $\approx$-representative in $B := \bigcup_i \Sigma^*(u_i)^\omega$. It remains to prune B to select unique representatives for each $\approx$-class.

It is easy to construct an ω-regular well-founded linear order on B. For every $w \in B$, let $p(w) \in \Sigma^*$ be the length-lexicographically smallest word such that w has period $p(w)$. Also let $t(w) \in \Sigma^*$ be the length-lexicographically smallest word so that $w = t(w) \cdot p(w)^\omega$. Define an order $\prec$ on B by $w \prec w'$ if $p(w)$ is length-lexicographically smaller than $p(w')$, or otherwise if $p(w) = p(w')$ and $t(w)$ is length-lexicographically smaller than $t(w')$. The ordering $\prec$ is ω-regular since it is FO-definable in terms of ω-regular relations. Finally, the required set of representatives may be defined as the set of $\prec$-minimal elements of every $\approx$-class; and an automaton for this set can be constructed from an automaton for $\approx$. ∎

This immediately yields an *injective* ω-automatic presentation from a given ω-automatic presentation which by Proposition 2.7 can be transformed into an automatic presentation of the structure. Thus we conclude that every countable ω-automatic structure is already automatic.

Corollary 2.8 *A countable structure is ω-automatic if and only if it is automatic. Transforming a presentation of one type into the other can be done effectively.*

Note that some of our technical results, in particular Lemmas 3.3 and 3.4, only require transitivity of the relation $\approx$ and do not use symmetry. Applying them to an ω-automatic linear order $\prec$ we get an interesting uncountable set of words of the form $\chi_S, S \subseteq \mathbb{N}$. For any two such words with $S \not\sim_{\mathrm{e}} T$, whether $\chi_S \prec \chi_T$ or not depends only on the first position $m \in S \triangle T$. Thus, $\prec$ behaves like the lexicographic order on such words.

4.1. Failure of Löwenheim-Skolem theorem for ω-automatic structures

While so far the area of automatic structures has mainly focused on individual structures, it is interesting to look at their theories as well. We note a consequence of our work for 'automatic model theory'.

An automatic version of the Downward Löwenheim-Skolem Theorem would say that every uncountable ω-automatic structure has a countable elementary substructure that is also ω-automatic. Unfortunately this is false since there is a first-order theory with an ω-automatic model but no countable ω-automatic model. Indeed, consider the first-order theory of atomless Boolean Algebras. Kuske and Lohrey [9] have observed that it has an uncountable ω-automatic model, namely $(\mathcal{P}(\mathbb{N}), \cap, \cup, \neg)/\sim_{\mathrm{e}}$. However, Khoussainov

et al. [7] show that the countable atomless Boolean algebra is not automatic and so, by Corollary 2.8, neither ω-automatic.

Here is the closest we can get to an automatic Downward Löwenheim-Skolem Theorem for ω-automatic structures.

Proposition 4.2. *Let $(D, \approx, \{R_i\}_{i\leq\omega})$ be an omega-automatic presentation of $\mathfrak{A}$ and let $\mathfrak{A}_{up}$ be its restriction to the ultimately periodic words of D. Then $\mathfrak{A}_{up}$ is a countable elementary substructure of $\mathfrak{A}$.*

Proof. Relying on the Tarski-Vaught criterion for elementary substructures we only need to show that for all first-order formulas $\varphi(\vec{x}, y)$ and elements $\vec{b}$ of $\mathfrak{A}_{\text{up}}$

$$\mathfrak{A} \models \exists y \varphi(\vec{b}, y) \quad \Rightarrow \quad \mathfrak{A}_{\text{up}} \models \exists y \varphi(\vec{b}, y) \ .$$

By Theorem 2.3 $\varphi(\vec{x}, y)$ defines an omega-regular relation and, similarly, since the parameters $\vec{b}$ are all ultimately periodic the set defined by $\varphi(\vec{b}, y)$ is omega-regular. Therefore, if it is non-empty, then it also contains an ultimately periodic word, which is precisely what we needed. ∎

This proof can be viewed as a model construction akin to a classical compactness proof. Indeed, starting with ultimately constant words and throwing in witnesses for all existential formulas satisfied in $\mathfrak{A}$ in each round one constructs an increasing sequence of substructures comprising ultimately periodic words of increasing period lengths. The union of these is closed under witnesses by construction. The argument is valid for relational structures with constants assuming that every constant is represented by an ultimately periodic word.

Future work It remains to be seen whether statements analogous to Theorem 2.4 and Corollary 2.8 also hold for automatic presentations over infinite trees.

Acknowledgment We thank the referees for detailed technical remarks and corrections.

References

[1] A. Blumensath. Automatic structures. Diploma thesis, RWTH-Aachen, 1999.
[2] A. Blumensath and E. Grädel. Automatic Structures. In *Proceedings of 15th IEEE Symposium on Logic in Computer Science LICS 2000*, pages 51–62, 2000.
[3] A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Comp. Sys.*, 37:641 – 674, 2004.
[4] G. Hjörth, B. Khoussainov, A. Montalban, and A. Nies. Borel structures. Manuscript, 2007.
[5] B.R. Hodgson. Décidabilité par automate fini. *Ann. sc. math. Québec*, 7(1):39–57, 1983.
[6] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC '94*, volume 960 of *LNCS*, pages 367–392. Springer-Verlag, 1995.
[7] B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: Richness and limitations. In *LICS*, pages 44–53. IEEE Comp. Soc., 2004.
[8] B. Khoussainov, S. Rubin, and F. Stephan. Definability and regularity in automatic structures. In *STACS '04*, volume 2996 of *LNCS*, pages 440–451, 2004.
[9] D. Kuske and M. Lohrey. First-order and counting theories of ω-automatic structures. In *FoSSaCS*, pages 322–336, 2006.
[10] D. Perrin and J.-E. Pin. Semigroups and automata on infinite words. In J. Fountain, editor, *Semigroups, Formal Languages and Groups*, NATO Advanced Study Institute, pages 49–72. Kluwer, 1995.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 397-408
www.stacs-conf.org

ON THE INDUCED MATCHING PROBLEM

IYAD A. KANJ [1], MICHAEL J. PELSMAJER [2], MARCUS SCHAEFER [1], AND GE XIA [3]

[1] DePaul University, Chicago, IL 60604, USA.
E-mail address: {ikanj,mschaefer}@cs.depaul.edu

[2] Illinois Institute of Technology, Chicago, IL 60616, USA.
E-mail address: pelsmajer@iit.edu

[3] Lafayette College, Easton PA 18042, USA.
E-mail address: gexia@cs.lafayette.edu

Abstract. We study extremal questions on induced matchings in several natural graph classes. We argue that these questions should be asked for twinless graphs, that is graphs not containing two vertices with the same neighborhood. We show that planar twinless graphs always contain an induced matching of size at least $n/40$ while there are planar twinless graphs that do not contain an induced matching of size $(n+10)/27$. We derive similar results for outerplanar graphs and graphs of bounded genus. These extremal results can be applied to the area of parameterized computation. For example, we show that the induced matching problem on planar graphs has a kernel of size at most $40k$ that is computable in linear time; this significantly improves the results of Moser and Sikdar (2007). We also show that we can decide in time $O(91^k + n)$ whether a planar graph contains an induced matching of size at least k.

Introduction

A matching in a graph is an *induced matching* if it occurs as an induced subgraph of the graph; we let $mim(G)$ denote the size of a maximum induced matching in G. Determining whether a graph has an induced matching of size at least k is NP-complete for general graphs and remains so even if restricted to bipartite graphs of maximum degree 4, planar bipartite graphs, 3-regular planar graphs (see [4] for a detailed history). Furthermore, approximating a maximum induced matching is difficult: the problem is APX-hard, even for $4r$-regular graphs [4, 14].

In terms of the parameterized complexity of the induced matching problem on general graphs, it is known that the problem is $W[1]$-*hard* [9]. Hence, according to the parameterized

1998 ACM Subject Classification: G.2.1, G.2.2, G.2.3.
Key words and phrases: Induced matching, bounded genus graphs, parameterized algorithms, kernel.
The work of the first author was supported in part by a DePaul University Competitive Research Grant. The work of the fourth author was supported in part by a Lafayette College research grant.

complexity hypothesis, it is unlikely that the problem is *fixed-parameter tractable*, that is, solvable in time $f(k)n^c$ for some constant c independent of k.

There are several classes of graphs for which the problem turns out to be polynomial time solvable, for example chordal graphs and outerplanar graphs (see [4] for a survey and [8] for the result on outerplanar graphs).

Very recently, Moser and Sidkar [8] considered the parameterized complexity of PLANAR-IM: finding an induced matching of size at least k in a planar graph. They showed that PLANAR-IM has a *linear problem kernel*, but left the constant in the kernel size undetermined. Their result automatically implies that the problem is fixed-parameter tractable.

In the current paper we take a combinatorial approach to the problem establishing lower and upper bounds on the size of induced matchings in certain graph classes. In particular, an application of our results to PLANAR-IM gives a significantly smaller problem kernel than the one given in [8]. We also apply the results to obtain a practical parameterized algorithm for PLANAR-IM that can be extended to graphs of bounded genus and could be used as a heuristic for general graphs.

Let us consider the induced matching problem from the point of view of extremal graph theory: How large can a graph be without containing an induced matching of size at least k? Of course, dense graphs such as K_n and $K_{n,n}$ pose an immediate obstacle to this question being meaningful, but they can easily be eliminated by restricting the maximum or the average degree of the graph. Indeed, for strong edge colorings the maximum degree restriction is popular: a *strong edge coloring* with k colors is a partition of the edge set into at most k induced matchings [12]. A greedy algorithm shows that graphs of maximum degree Δ have a strong edge chromatic number of at most $2\Delta(\Delta-1)+1$, and, of course, Δ is an immediate lower bound. If we are only interested in a large induced matching though, perhaps we need not restrict the maximum degree. On the other hand, bounding only the average degree of a graph allows pathological examples such as $K_{1,n}$, which has average degree less than 2 but only a single-edge induced matching. This example illustrates another obstacle to a large induced matching: *twins*. Two vertices u and v are said to be twins if $N(u) = N(v)$. Obviously, at most one of u and v can be an endpoint of an edge in an induced matching and if one of them can, either can. Thus, from the extremal point of view (and since they can be easily recognized and eliminated) we should study the induced matching problem on graphs without twins. Twinlessness does not allow us to drop the bounded average degree requirement however, as shown by removing a perfect matching from $K_{n,n}$, which yields a twinless graph with a maximum induced matching of size 2.

We begin by studying twinless graphs of bounded average degree. Those graphs might still not have large induced matchings since they could contain very dense subgraphs (Remark 1.3 elaborates on this point). One way of dealing with this problem is to extend the average degree requirement to all subgraphs. In Section 1 we see that a slightly weaker condition is sufficient, namely a bound on the chromatic number of the graph. We can show that a graph of average degree at most d and chromatic number at most k contains an induced matching of size $\Theta(n^{1/(d+1)})$.

While we cannot expect to substantially improve the dependency on the average degree of this result in general (see Remark 1.2), we do investigate the case of planar graphs and graphs of bounded genus, for which we can show the existence of induced matchings of linear size. Indeed, a planar twinless graph always contains an induced matching of size $n/40$. We also know that this bound cannot be improved beyond $(n+10)/27$ (Remark 2.10). Planar graphs and graphs of bounded genus are discussed in Section 2.

We next investigate the case of outerplanar graphs: an outerplanar graph of minimum degree 2 always contains an induced matching of size $n/7$ (even without assuming twinlessness), and this result is tight (Section 3). Our bounds fit in with a long series of combinatorial results on finding sharp bounds on the size of induced structures in subclasses of planar graphs (see for example [5, 11, 1, 10]).

We also use our combinatorial results to obtain fixed-parameter algorithms for the induced matching problem. For example, we show that PLANAR-IM can be solved in time $O(91^k + n)$ by a very practical algorithm, while—on the more theoretical side—there is an algorithm deciding it in time $O(2^{159\sqrt{k}} + n)$ using the Lipton-Tarjan [7] separator theorem. Both results easily extend to graphs of bounded genus.

For graph-theoretic terminology we refer the reader to West [13]. For background on parameterized complexity, we recommend Downey and Fellows [3].

1. Induced matchings in graphs of bounded average degree

We can show that twinless graphs of bounded average degree and bounded chromatic number contain large induced matchings. At the core of the proof is a combinatorial result due to Füredi and Tuza [6, Theorem 9.13] on systems of strong representatives. For lack of space, we omit the details.

Theorem 1.1. *A twinless graph G with $\chi(G) \leq k$ and average degree at most d must contain an induced matching of size at least*

$$\left(\frac{d}{2} \left(\frac{n-1}{2k(d+1)} \right)^{1/(d+1)} - (d+1) \right) / (k-1)$$

which is $\Theta(n^{1/(d+1)})$ where $n = |V(G)|$.

Remark 1.2. Consider the following bipartite graph: take a set A of ℓ vertices, and for every $d/2$ element subset of A create a new vertex and connect it to the vertices of the subset.

This graph has $n = \ell + \binom{\ell}{d/2}$ vertices, and its largest induced matching has size $\ell/(d/2)$. Moreover, its average degree is $2 \cdot \frac{d}{2}\binom{\ell}{d/2} / \left(\ell + \binom{\ell}{d/2}\right) \leq d$. For d fixed, $\ell/(d/2)$ is of order $n^{2/d}$, which shows that the bound of the theorem (while not being tight) has the right form.

Remark 1.3. The preceding example can be extended to show that bounding the chromatic number is necessary: take the graph as constructed in the previous remark and add all edges between the ℓ vertices of A. Assuming $d \geq 4$, this gives a graph of average degree at most $d + 2$. However, the largest induced matching in this graph has size 1.

2. Planar graphs and graphs of bounded genus

2.1. Matchings and induced matchings

To find large induced matchings in graphs we often proceed in two steps: we first find a large matching in the graph and then turn it into an induced matching. To make this approach work, we need assumptions on the graph: to obtain a large matching, we assume

an upper bound on $\alpha(G)$, the size of the largest independent set in G. To turn the matching into an induced matching, we assume that the graph is twinless and all minors of G have a large independent set.

Lemma 2.1. *A graph G with $\alpha(G) \leq \alpha n$ contains a matching of size at least $(1-\alpha)n/2$, where $n = n(G)$.*

Proof. Let $M \subseteq E$ be a maximal matching in G on vertex set $V(M)$. Then $I = V - V(M)$ is an independent set. By assumption, $|I| \leq \alpha n$. Adding $|V(M)|$ to either side gives us $n \leq \alpha n + |V(M)|$, and, therefore, $|V(M)| \geq (1-\alpha)n$. ∎

Lemma 2.2. *Assume that any minor $H \preceq G$ of a graph G fulfills $\alpha(H) \geq \alpha\, n(H)$. Then any matching M in G contains an induced matching in G of size at least $\alpha|M|$.*

Proof. Remove all vertices not in $V(M)$ and contract the edges of M (removing duplicate edges). The resulting graph is a minor of G, and, by assumption, has an independent set of size $\alpha|M|$. The edges in M which were contracted to the vertices in the independent set, form an induced matching in G. ∎

By this lemma a matching of size k in a planar graph contains an induced matching of size $k/4$. In [2] the authors show that a 3-connected planar graph contains a matching of size at least $(n+4)/3$, which allows us to draw the following conclusion.

Corollary 2.3. *A 3-connected planar graph contains an induced matching of size $(n+4)/12$.*

To apply the two lemmas to planar graphs and graphs of bounded genus we need some generalizations of Euler's theorem to hypergraphs. We say a *hypergraph $\mathcal{H}$ is embeddable in a surface* if the bipartite incidence graph obtained from $\mathcal{H}$ by replacing each of its edges by a vertex adjacent to all the vertices in the edge is embeddable in that surface.

Lemma 2.4. *A hypergraph of genus at most g on n vertices has at most $2n + 4g - 4$ edges containing at least three vertices, unless $n = 1$ and $g = 0$.*

If $\mathcal{H}$ is a hypergraph of genus g such that all edges have size 2, we can take the associated bigraph G of genus g and contract away all the the vertices that correspond to edges of $\mathcal{H}$. This produces a graph of genus g with $|V(\mathcal{H})|$ vertices and $|E(\mathcal{H})|$ edges, to which we may apply the following consequence of Euler's Theorem.

Lemma 2.5 (Euler)**.** *A graph of genus g on n vertices contains at most $3n + 6g - 6$ edges if $n \geq 2$.*

By splitting edges of a hypergraph into those of size at least three, those of size two, and those that contain a single vertex, we can derive the following.

Lemma 2.6. *A hypergraph of genus at most g on n vertices has at most $6n + 10g - 9$ edges if $n \geq 2$.*

We are now ready to give a lower bound on the size of induced matchings in twinless graphs of bounded genus. This includes the planar case, but in the next section we will give an improved bound for that case. We need a result due to Heawood [13] that states that a graph of genus at most g can be colored using at most $(7 + \sqrt{1 + 48g})/2$ colors. The statement remains true for the plane case, $g = 0$, by virtue of the Four-Color Theorem.

Theorem 2.7. *A twinless graph of genus at most g contains an induced matching of size at least $(n - 10g)/(49 + 7\sqrt{1 + 48g})$, where n is the number of vertices of the graph.*

Proof. Let G be a twinless graph of genus at most g, and assume temporarily that G does not contain any isolated vertex. Let $M \subseteq E$ be a maximal matching in G on vertex set $V(M)$. Then $I = V - V(M)$ is an independent set. Let $\mathcal{H}$ be the hypergraph with vertex set $V(M)$ and edges $N(v)$, $v \in I$. Then $\mathcal{H}$ is a hypergraph of genus at most g (as its bipartite incidence graph is a subgraph of G), and by Lemma 2.6, has at most $6|V(M)|+10g-9$ edges (note that we can assume $|V(M)| \geq 2$ since otherwise G consists of a single vertex, in which case there is nothing to prove). As G contains no twins, each edge of $\mathcal{H}$ uniquely corresponds to a vertex in I, so $|I| \leq 6|V(M)| + 10g - 9$ and, therefore, $|V(M)| \geq (|V| - 10g + 9)/7$. The original graph might have contained at most one isolated vertex (since it is twinless), so $|V(M)| \geq (n - 10g)/7$ and G has a matching of size at least $(n - 10g)/14$.

By Heawood's theorem and the Four-Color Theorem, a graph of genus at most g can be colored using at most $(7+\sqrt{1+48g})/2$ colors. Hence, G and any of its minors always contain independent sets on a $2/(7 + \sqrt{1+48g})$ fraction of their vertices. Then by Lemma 2.2, G has an induced matching of size at least $2(n - 10g)/[14(7 + \sqrt{1+48g})] = (n - 10g)/(49 + 7\sqrt{1+48g})$. ∎

A simple consequence of Theorem 2.7 not involving the concept of twinlessness is the following:

Corollary 2.8. *A planar graph of minimum degree at least* 3 *contains an induced matching of size at least* $(n+8)/20$*, where* n *is the number of vertices of the graph.*

Proof. Since the graph has minimum degree at least 3 it cannot contain degree 1 and 2 vertices. Then by Lemma 2.4, the hypergraph constructed in the proof of Theorem 2.7 (for $g = 0$) contains at most $2|V(M)| - 4$ edges. However, it is now possible that more than one vertex in the independent set results in the same edge of the hypergraph. However, there can be at most two vertices sharing the same neighborhood, since a planar graph does not contain a $K_{3,3}$. Therefore, the size of the independent set is at most $4|V(M)| - 8$, and thus the graph contains a matching of size at least $(n+8)/5$. Using Lemma 2.2, it can be turned into an induced matching of size at least $(n + 8)/20$. ∎

Theorem 2.7 implies that a planar twinless graph always contains an induced matching of size $n/56$. This lower bound can still be improved as shown in the following theorem.

Theorem 2.9. *A twinless planar graph contains an induced matching of size at least* $n/40$*, where* n *is the number of vertices of the graph.*

Remark 2.10. We do not have a matching upper bound to complement Theorem 2.9, but we can get close. We can construct a graph whose largest induced matching has size $(n + 10)/27$.

3. Induced matchings in outerplanar graphs

The main result of this section is that a nontrivial connected outerplanar graph with minimum degree 2 has an induced matching of size $\lceil \frac{n}{7} \rceil$. This result is sharp, as will be seen later. We first consider a special case, which will arise later in the proof of the main result. We refer the reader to [13] for the terminology on the block decomposition of a graph.

Lemma 3.1. *Suppose that* G *is a connected graph for which the block-cutpoint tree is a path and all blocks are triangles or cut-edges; or, equivalently,* G *is the union of a path of*

length $\ell \geq 1$ and at most ℓ triangles, with each edge of the path in at most one triangle, and exactly one edge of each triangle in the path. If $2 \leq n(G) \leq 5$ then $mim(G) \geq \lceil \frac{n(G)+1}{6} \rceil$ and if $n(G) \geq 6$ then $mim(G) \geq \lceil \frac{n(G)+3}{6} \rceil$.

Corollary 3.2. *Let G be a 2-connected outerplanar graph with exactly one non-leaf face, such that every leaf face is a 3-face. Then for any vertex v, $mim(G - v) \geq \lceil \frac{n(G)}{6} \rceil$.*

To prove the main result of this section, we use induction after separating the graph into components (by removing vertices that form a certain cut in the graph). To apply the inductive statement, each of these components must have minimum degree 2. This, however, may not be true after the removal of the cut-set from the graph. We next define an operation, called the *patching operation*, that patches each of these components so that its minimum degree is 2.

Definition 3.3. Let H be an outerplanar graph with $n(H) \geq 4$ and with at most two degree 1 vertices. We define an operation that can be applied to H, called the *patching operation*, to obtain a graph H' as follows.

(a) If there is no degree 1 vertex in H let $H' = H$.
(b) If there is exactly one degree 1 vertex u in H, let u' be its neighbor. If $\deg_H(u') \geq 3$, let $H' = H - u$. Otherwise ($\deg_H(u') = 2$), let v be the other neighbor of u'. Let v' be a vertex after v on the boundary walk in $H - \{u, u'\}$. Let $H' = (H - u) + u'v'$.
(c) If there are exactly two degree 1 vertices u and v in H, let u' be the neighbor of u and v' be the neighbor of v. Remove u from H and add the edge $u'v$. Let H' be the resulting graph.

Proposition 3.4. *Let H be an outerplanar graph with $n(H) \geq 4$ and with at most two degree 1 vertices. Moreover, when H has exactly two degree 1 vertices u and v, then adding a path from u to v leaves H outerplanar. Let H' be the graph resulting from the application of the patching operation to H. Then H' is an outerplanar graph such that: (1) the minimum degree of H' is 2, (2) $n(H') \geq n(H) - 1$, and (3) $mim(H) \geq mim(H')$.*

Theorem 3.5. *A nontrivial connected outerplanar graph G of minimum degree 2 has an induced matching of size $\lceil \frac{n}{7} \rceil$.*

Proof. Clearly the statement is true if $3 \leq n \leq 7$. Therefore, we may assume in the remainder of the proof that $n \geq 8$, and that, inductively, the statement is true for any graph with fewer than n vertices.

Let u be a cut-point in G which is in at most one non-leaf block. Let $B_1, \cdots, B_\ell$ be all the leaf blocks containing u, let $B_0 = G - \bigcup_{i=1}^{\ell} [V(B_i) - u]$, and let $n_i = n(B_i)$, for $i = 0, \cdots, \ell$. If G has no cut-points, let u be any vertex in G, and let $B_0 = G$.

Let B_i, where $i \in \{1, \cdots, \ell\}$ be a block such that $n_i \geq 7$. Let B_i' be the block obtained from B_i by deleting the chord of each 3-face of B_i. Suppose that B_i' is not a cycle. Clearly, any leaf face in B_i' must be of length at least 4.

Suppose that B_i' has a leaf face of length at least 6, with boundary $F = (u_1, \ldots, u_r, u_1)$ such that $u_1 u_r$ is a chord and $u_1 \neq u$. Let $H = G - \{u_1, u_2, u_3, u_4, u_5\}$, and note that none of the vertices in H is a cut-point in G. Therefore, H is an outerplanar graph with at most two degree 1 vertices. Apply the patching operation to H to obtain a graph H'. Then H' is a connected outerplanar graph with minimum degree two. Inductively, we have $mim(H') \geq \lceil \frac{n(H')}{7} \rceil$. Since $n(H') \geq n(H) - 1$ and $mim(H) \geq mim(H')$ by Proposition 3.4,

we have $mim(H) \geq \lceil \frac{n(H)-1}{7} \rceil = \lceil \frac{n(G)-6}{7} \rceil$. A maximum induced matching in H plus edge u_2u_3 is an induced matching in G, because any edge of $E(B_i) - E(B_i')$ incident to u_2 or u_3 has at its other endpoint u_1, u_4, or u_5, by the construction of B_i' and F. We conclude that $mim(G) \geq \lceil \frac{n(G)-6}{7} \rceil + 1 = \lceil \frac{n(G)+1}{7} \rceil$, which suffices.

If B_i' contains a leaf face $F = (u_1, \cdots, u_r, u_1)$ with $r = 4$ or $r = 5$, and such that $u_1 \neq u$ and $u_r \neq u$, then similar to the above, we let $H = G - \{u_1, \cdots, u_r\}$. Again note that none of the vertices in H is a cut-point in G. Using the same analysis as in the above paragraph, we obtain $mim(G) \geq \lceil \frac{n(G)+1}{7} \rceil$.

Assuming that $n_i \geq 7$ and that B_i' is not a cycle, it follows now that every leaf face in B_i' has length 4 or 5 and is incident to the cut-point u in G. Therefore, B_i' has exactly two leaf faces that contain u, and each of length 4 or 5. Let $F = (u_1, \cdots, u_r, u_1)$ and $F' = (u_1', \cdots, u_s', u_1')$ where $r, s \in \{4, 5\}$, $u = u_1 = u_1'$, and u_1u_r and $u_1'u_s'$ are chords. Note that it is possible that $u_r = u_s'$. Let H be the graph obtained from B_i by removing the vertices in $F \cup F'$; then H is a path so it has at most two vertices of degree 1. If $n(H) \geq 1$ then the edges u_2u_3 and $u_2'u_3'$ give an induced matching in B_i of size 2. Since $n_i \leq 10$, B_i has a matching M_i of size at least $\lceil \frac{n_i+4}{7} \rceil$. If $n(H)$ is 2 or 3, then H has a maximum induced matching of size at least 1, which together with edges u_2u_3 and $u_2'u_3'$ give an induced matching in B_i of size 3. Since in this case $n_i \leq 12$, we conclude that B_i has an induced matching M_i of at least $\lceil \frac{n_i}{6} \rceil$. Moreover, no edge of M_i is incident on the cut-point u of G. Now if $n(H) \geq 4$, we apply the patching operation to H to obtain an outerplanar graph of minimum degree two. Inductively, $mim(H') \geq \lceil \frac{n(H')}{7} \rceil$, and hence $mim(H) \geq \lceil \frac{n(H)-1}{7} \rceil$. Now any induced matching in H plus edges u_2u_3 and $u_2'u_3'$ gives an induced matching M_i in B_i such that none of the edges in M_i is incident on u. It follows that $mim(G) \geq 2 + mim(H) \geq 2 + \lceil \frac{n(H)-1}{7} \rceil \geq 2 + \lceil \frac{n_i-9-1}{7} \rceil \geq \lceil \frac{n_i+4}{7} \rceil$. Therefore, in this case B_i contains an induced matching M_i, none of its edges is incident on u, of size at least $\lceil \frac{n_i+4}{7} \rceil$.

Now, for any $i \in \{1, \cdots, \ell\}$ we have the following:

If $n_i \leq 6$, then clearly B_i contains an induced matching M_i, none of its edges incident on u, of size at least $\lceil \frac{n_i}{6} \rceil$. Simply let M_i be any edge in B_i that is not incident on u.

If $n_i \geq 7$ and B_i' is a cycle, then B_i satisfies the conditions of Corollary 3.2, and B_i has an induced matching M_i of size at least $\lceil \frac{n_i}{6} \rceil$, none of its edges is incident on u (by choosing $v = u$ in Corollary 3.2).

If $n_i \geq 7$, and B_i' is not a cycle, then from the above discussion, B_i has an induced matching of size at least $min\{\lceil \frac{n_i+4}{7} \rceil, \lceil \frac{n_i}{6} \rceil\}$.

Let $M = \bigcup_{i=1}^{\ell} M_i$. Let $H = B_0 - u$ and note that H has at most two degree 1 vertices. If $n(H) \leq 3$, then clearly $mim(H) \geq \lceil \frac{n_0}{6} \rceil$. If $n(H) \geq 4$, apply the patching operation to H to obtain an outerplanar graph H' of minimum degree 2. Now by applying the inductive statement to H', we get $mim(B_0) \geq \lceil \frac{n_0-2}{7} \rceil$. Let M_0 be a maximum induced matching in $B_0 - u$, and note that since none of the induced matching edges in $M \cup M_0$ is incident on u, $M \cup M_0$ is an induced matching in G.

If G has no cut-points, then G is 2-connected and we let $B_1 = G$. In this case we have $mim(G) \geq min\{\lceil \frac{n(G)+4}{7} \rceil, \lceil \frac{n(G)}{6} \rceil\} \geq \lceil \frac{n(G)}{7} \rceil$.

Now we can assume that $\ell \geq 1$. Note that in this case we have $n_0 + n_1 + \cdots + n_\ell = n + \ell$.

If at least one block B_i has $|M_i| \geq \lceil \frac{n_i+4}{7} \rceil$, then by using $\lceil \frac{n_i}{7} \rceil$ as a lower bound on the size of the matching in each block B_j where $j \in \{1, \cdots, \ell\}$ and $j \neq i$, we get:

$$|M \cup M_0| \geq \sum_{j=1, j\neq i}^{\ell} \lceil \frac{n_j}{7} \rceil + \lceil \frac{n_i + 4}{7} \rceil + \lceil \frac{n_0 - 2}{7} \rceil \geq \lceil \frac{n + 2 + \ell}{7} \rceil \geq \lceil \frac{n}{7} \rceil.$$

Otherwise, we can use $\lceil \frac{n_i}{6} \rceil$ as a lower bound on the size of each block B_i where $i \in \{1, \cdots, \ell\}$. If $\ell \geq 2$, we have:

$$|M \cup M_0| \geq \sum_{i=1}^{\ell} \lceil \frac{n_i}{6} \rceil + \lceil \frac{n_0 - 2}{7} \rceil \geq \sum_{i=1}^{\ell} \lceil \frac{n_i}{7} \rceil + \lceil \frac{n_0 - 2}{7} \rceil \geq \lceil \frac{n + \ell - 2}{7} \rceil \geq \lceil \frac{n}{7} \rceil.$$

If $\ell = 1$ and $n_1 \leq 5$, by picking M to be any edge that is not incident on u in block B_1, we get:

$$|M \cup M_0| \geq 1 + \lceil \frac{n_0 - 2}{7} \rceil = \lceil \frac{n_0 + 5}{7} \rceil \geq \lceil \frac{n}{7} \rceil.$$

If $\ell = 1$ and $n_1 \geq 6$, we have:

$$\begin{aligned} |M \cup M_0| &\geq \lceil \frac{n_1}{6} \rceil + \lceil \frac{n_0 - 2}{7} \rceil \geq \lceil \frac{7n_1 + 6n_0 - 12}{42} \rceil = \lceil \frac{6(n_1 + n_0) + n_1 - 12}{42} \rceil \\ &\geq \lceil \frac{6n + 6 + n_1 - 12}{42} \rceil \geq \lceil \frac{n}{7} \rceil. \end{aligned}$$

This completes the induction and the proof. ∎

Figure 1 shows an example of a graph in which the size of the maximum induced matching is exactly $\lceil n/7 \rceil$. A graph in this family consists of a cycle of length 2ℓ ($\ell \geq 3$) with ℓ gadgets attached as indicated in the figure. The total number of vertices in this graph is 7ℓ, and it is easy to verify that the maximum induced matching has size exactly ℓ.

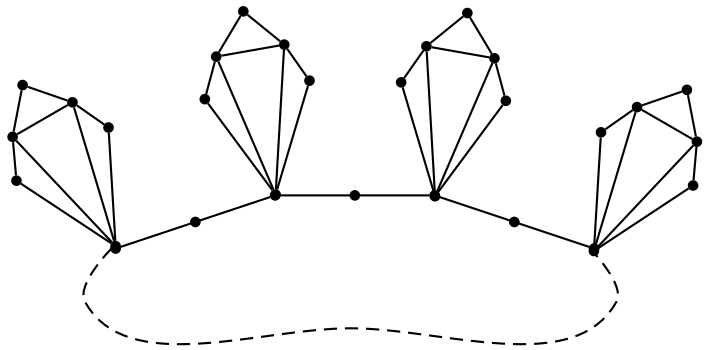


Figure 1: An illustration of a family of outerplanar graphs for which the lower bound on the size of an induced matching is tight.

4. Applications to parameterized computation

In this section we apply our previous results to obtain parameterized algorithms for IM on graphs of bounded genus. Let (G, k) be an instance of IM where G has n vertices and genus g for some integer constant $g \geq 0$.

4.1. A problem kernel

We first show how to kernelize the instance (G, k) when G is planar (i.e., for the case $g = 0$). We then extend the results to graphs with genus g for any integer constant $g > 0$.

Theorem 2.9 shows that any twinless planar graph on n vertices has an induced matching of at least $n/40$ edges. Observing that if u is a vertex in G that has a twin then $mim(G) = mim(G - u)$, by repeatedly removing every vertex in G with a twin, we end up with a twinless graph G' such that G has an induced matching of size k if and only if G' does. If $k \leq n(G')/40$ then the instance (G', k) of IM can be accepted; otherwise, the instance (G', k) is a kernel of (G, k) with $n(G') \leq 40k$, and we can work on (G', k).

Therefore, our task amounts to reducing the graph G to the twinless graph G'. We describe next how this can be done in linear time.

Assume that G is given by its adjacency list and that the vertices in G are labeled by the integers $1, \ldots, n$. We can further assume that the neighbors of every vertex appear in the adjacency list in increasing order. If this is not the case, we create the desired adjacency list by enumerating the vertices in increasing order, and inserting each vertex in the neighborhood list of each of its adjacent vertices. This can be easily done in $O(n)$ time.

For every vertex v of degree d, we associate a d-digit number $x_v = v_1 \cdots v_d$, where $v_1, \ldots, v_d$ are the neighbors of v in the order they appear in the adjacency list of v (i.e., in increasing order). We perform a radix sort on the numbers associated with the vertices of G using only the first three or less (leftmost) digits of these numbers. Since each digit is a number in the range $1 \ldots n$, and there are at most $O(n)$ numbers (twice the number of edges in the planar graph), radix sort takes $O(n)$ time. Let π be this sorted list. Observe that two vertices u and v are twins if and only if $x_u = x_v$. Moreover, since the graph is planar, and hence does not contain the complete bipartite graph $K_{r,r}$ for any integer $r \geq 3$, any twin vertices of degree at least 3 must have their numbers adjacent in π (otherwise there would be at least 3 vertices with the same neighborhood). Therefore, we can recognize the twins in G as follows. Process the numbers in π in order: Let x_u and x_v be two adjacent numbers in π, and assume that x_u appears before x_v. We check whether u and v are twins by comparing the corresponding digits of x_u and x_v. If u and v are twins, we mark u. When we have finished this process, we remove all marked vertices from the graph. We let G' be the resulting graph. Since for each number x_u in π we spend time proportional to the number of digits in x_u and that of the number appearing next to x_u in π, the running time is proportional to the sum of the degrees of the vertices in G, which is $O(n)$.

Theorem 4.1. *Let (G, k) be an instance of* IM *where G is a planar graph on n vertices. Then in $O(n)$ time we can compute an instance (G', k') where (G', k') is a kernel of (G, k), $k' \leq k$, and such that either $n(G') \geq 40k'$ and we can accept the instance (G, k), or $n(G') < 40k'$.*

The above theorem gives a kernel of size $40k$ for Planar-IM, and is a significant improvement on the results in [8] where a kernel of size $O(k)$ was derived without the constant in the asymptotic notation being specified. The above results give a concrete value for the bound on the kernel size. Moreover, this value is moderately small and the analysis techniques are much simpler when compared to the technique of decomposing a planar graph into regions used in [8].

The same technique can be used to eliminate twin vertices from a graph with genus g. Using Euler's formula on $K_{r,r}$ with the fact that faces in an embedded bipartite graph have length at least 4, it can be easily shown that:

Proposition 4.2. *A graph with genus g does not contain the complete bipartite graph $K_{r,r}$ for any $r > 2 + 2\sqrt{g}$.*

Using Theorem 2.7 and Proposition 4.2, Theorem 4.1 can now be generalized to graphs with bounded genus.

Theorem 4.3. *Let (G, k) be an instance of* IM *where G is a graph on n vertices with genus g. Then in $O(gn)$ time we can compute an instance (G', k') where (G', k') is a kernel of (G, k), $k' \leq k$, and such that either $n(G') \geq (49 + 7\sqrt{1 + 48g})k' + 10g$ and we can accept the instance (G, k), or $n(G') < (49 + 7\sqrt{1 + 48g})k' + 10g$.*

4.2. Parameterized algorithms for IM on graphs with bounded genus

We again begin with the planar case. Assume that we have an instance (G, k) of Planar-IM. By Theorem 4.1, we can assume that after an $O(n)$ preprocessing time, the number of vertices n in G satisfies $n \leq 40k$. We will show how to design a parameterized algorithm for the Planar-IM problem. Our algorithm is a bounded-search-tree algorithm that uses the Lipton-Tarjan separator theorem [7]. Our results answer an open question posed by [8] of whether a bounded-search-tree algorithm exists for Planar-IM. We also show at the end of this section how these results can be extended to bounded genus graphs.

Theorem 4.4 ([7])**.** *Given a planar graph $G = (V, E)$ on n vertices, there is a linear time algorithm that partitions V into vertex-sets A, B, S such that:*

1. *$|A|, |B| \leq 2n/3$;*
2. *$|S| \leq \sqrt{8n}$; and*
3. *S separates A and B, i.e. there is no edge between a vertex in A and and a vertex in B.*

Given an instance (G, k) of Planar-IM, where $G = (V, E)$ and $|V| = n$, we partition V into vertex-sets A, B, S according to the Lipton-Tarjan theorem. Let G_A, G_B, and G_S be the subgraphs of G induced by the vertices in A, B, and S, respectively. The idea is simple: separate the graph by enumerating a possible status for the vertices in S, and then use a divide-and-conquer approach. However, special care needs to be taken when enumerating the vertices in S as this enumeration is not straightforward. We outline the general approach below

Each vertex u in S is either an endpoint of an edge in the induced matching or not. Therefore, we assign each vertex u two possible statuses: status 0 if u is an endpoint of an edge in the induced matching and 1 if it is not. Suppose that we have assigned a status to every vertex u in S. If the assigned status to u is 0, we simply remove u (and its incident edges) from G. If the assigned status to u is 1 and there is an edge uu' where $u' \in S$ and the status assigned to u' is 1, then uu' has to be an edge in the induced matching if our enumeration is correct. Therefore, we can add uu' to the matching and remove all the neighbors of u and u' from G. If the assigned status to u is 1, and no vertex $u' \in S$ exists such that the assigned status to u' is 1, then we further assign u two statuses: status 1_A if u is matched to a vertex in G_A in the induced matching, and status 1_B if u is matched to a vertex in G_B. In the former case, we add u to G_A and remove all its neighbors in G_B, and in the latter case, we add u to B and remove all its neighbors in G_A.

After assigning each vertex in S a status from $\{0, 1_A, 1_B\}$, and updating the graph according to the above description, G_A and G_B are separated, and we can recurse on them

to compute an induced matching M_A of G_A and M_B of G_B. We then return $M_A \cup M_B$ plus all the edges uu' where $u, u' \in S$, and the assigned status to u and u' is 1. Note that since our enumeration might be incorrect, the returned set of edges may not correspond to an induced matching. Therefore, we will need to verify that the returned set corresponds to an induced matching before returning it.

If there is an induced matching of at least k edges in G, then it is not difficult to see that at least one enumeration will return such an induced matching. Otherwise, no enumeration can find an induced matching of at least k edges, and we reject the instance.

Finally, note that in the recursive calls, some of the vertices in G_A and G_B may have already been assigned the status 1, and we need to respect the assigned statuses in any possible future enumeration of those vertices in G_A and G_B.

A standard analysis shows that the running time of the algorithm is $O(2^{25\sqrt{n}})$. Noting that $n \leq 40k$, we have the following theorem:

Theorem 4.5. *In time $O(2^{159\sqrt{k}} + n)$, it can be determined whether a planar graph on n vertices has an induced matching of at least k edges.*

The above results can be extended to bounded genus graphs.

Theorem 4.6. *Let G be a graph on n vertices with genus g. In time $O(2^{O(\sqrt{gk})} + n)$, it can be determined whether G has an induced matching of at least k edges.*

Due to the large constant in the exponent of the running time of the above algorithms, it is clear that these algorithms are far from being practical. We shall present in the next section more practical parameterized algorithms for IM on bounded genus graphs.

5. Practical algorithms for IM on graphs of bounded genus

We start with the planar case. Let (G, k) be an instance of Planar-IM where G has n vertices. By Theorem 4.1, we can assume that after an $O(n)$ preprocessing time, the number of vertices n in G satisfies $n \leq 40k$.

Let M be a maximal matching in G and let $I = V(G) - V(M)$. If $V(M)$ contains more than $8k$ vertices, then by contracting each edge of M in $G_M = G(V(M))$ then applying the Four-Color Theorem to G_M, we conclude that G_M, and hence G, has an induced matching of at least k edges, and we can accept the instance (G, k). Assume that $V(M) < 8k$.

The algorithm will look for a set of exactly k edges that form an induced matching. These edges will have at most $2k$ endpoints in $V(M)$. Therefore, we start by enumerating every subset $S \subseteq V(M)$ of size at most $2k$. There are at most $\sum_{i=0}^{2k} \binom{8k}{i}$ such subsets. Let S be such a subset. We work under the assumption that every vertex in S is an endpoint of an edge in the induced matching until we either find the desired induced matching, or this assumption turns out to be false. In the latter case we enumerate the next subset S.

If two vertices u and v in S are adjacent, then uv must be an edge in the induced matching; therefore, in this case we include uv, remove every neighbor of u and v from G, and reduce k by 1. After we have included (in the induced matching) every edge both of whose endpoints are in S, every remaining vertex in S must be matched with a vertex in I. Observe that if there is a vertex $w \in I$ that is adjacent to at least two vertices in S, then none of the edges joining w to S is in the induced matching. Hence, w could not be an endpoint to an edge in the matching, and w can be removed from I. After removing every such vertex w from I, each remaining vertex in I is adjacent to at most one vertex in

S. Now if our original choice of the set S was correct, then by choosing a neighbor in I for every vertex in S, we should obtain an induced matching in G of size k. If such a choice is not possible (for example, a vertex in S does not have a neighbor in I), or the total number of edges in the induced matching at the end of this process is less than k, then our choice of S was incorrect, and we enumerate the next subset S of $V(M)$ of size at most $2k$. After we have enumerated all subsets of $V(M)$ of size at most $2k$, either we have found an induced matching of at least k edges, or no such matching exists. Noting that there are at most $\sum_{i=0}^{2k} \binom{8k}{i} \leq (2k+1)\binom{8k}{2k}$ such subsets, and that the number of vertices in G is $O(k)$, we have the following theorem:

Theorem 5.1. PLANAR-IM *can be solved in* $O(\binom{8k}{2k}k^2 + n) = O(91^k + n)$ *time.*

This algorithm is more practical for small values of the parameter k than the one described previously. We can generalize the result to bounded genus graphs:

Theorem 5.2. *The* IM *problem on graphs with* n *vertices and genus* g *can be solved in* $O(\binom{(7+\sqrt{1+48g})k}{2k}k^2 + n)$ *time.*

References

[1] Noga Alon. Problems and results in extremal combinatorics. I. *Discrete Math.*, 273(1-3):31–53, 2003. EuroComb'01 (Barcelona).

[2] Therese Biedl, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer, and Stephen G. Kobourov. Tight bounds on maximal and maximum matchings. *Discrete Math.*, 285(1-3):7–15, 2004.

[3] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. Parameterized complexity: a framework for systematically confronting computational intractability. In *Contemporary trends in discrete mathematics (Štiřín Castle, 1997)*, volume 49 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 49–99. Amer. Math. Soc., Providence, RI, 1999.

[4] William Duckworth, David F. Manlove, and Michele Zito. On the approximability of the maximum induced matching problem. *J. Discrete Algorithms*, 3(1):79–91, 2005.

[5] Kiyoshi Hosono. Induced forests in trees and outerplanar graphs. *Proc. Fac. Sci. Tokai Univ.*, 25:27–29, 1990.

[6] Stasys Jukna. *Extremal combinatorics: With applications in computer science*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2001.

[7] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[8] Hannes Moser and Somnath Sikdar. The parameterized complexity of the induced matching problem in planar graphs. In *Proceedings of the 2007 International Frontiers of Algorithmics Workshop*, Lecture Notes in Comput. Sci., Berlin, 2007, to appear. Springer.

[9] Hannes Moser and Dimitrios M. Thilikos. Parameterized complexity of finding regular induced subgraphs. In *Proceedings of the Second Workshop on Algorithms and Complexity in Durham*, volume 7 of *Texts in Algorithmics*, pages 107–118. King's College, London, 2006.

[10] Michael J. Pelsmajer. Maximum induced linear forests in outerplanar graphs. *Graphs Combin.*, 20(1):121–129, 2004.

[11] Mohammad R. Salavatipour. Large induced forests in triangle-free planar graphs. *Graphs Combin.*, 22(1):113–126, 2006.

[12] Douglas B. West. Open problems - strong edge coloring. `http://www.math.uiuc.edu/~west/openp/strongedge.html` (accessed August 11th, 2007).

[13] Douglas B. West. *Introduction to graph theory*. Prentice Hall Inc., Upper Saddle River, NJ, 1996.

[14] Michele Zito. Induced matchings in regular graphs and trees. In *Graph-theoretic concepts in computer science (Ascona, 1999)*, volume 1665 of *Lect. Notes in Comput. Science*, pp. 89-100. Springer, 1999.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 409-420
www.stacs-conf.org

ON GEOMETRIC SPANNERS OF EUCLIDEAN AND UNIT DISK GRAPHS

IYAD A. KANJ AND LJUBOMIR PERKOVIĆ

DePaul University, Chicago, IL 60604, USA.

E-mail address: {ikanj,lperkovic}@cs.depaul.edu

Abstract. We consider the problem of constructing bounded-degree planar geometric spanners of Euclidean and unit-disk graphs. It is well known that the Delaunay subgraph is a planar geometric spanner with stretch factor $C_{del} \approx 2.42$; however, its degree may not be bounded. Our first result is a very simple linear time algorithm for constructing a subgraph of the Delaunay graph with stretch factor $\rho = 1 + 2\pi(k \cos \frac{\pi}{k})^{-1}$ and degree bounded by k, for any integer parameter $k \geq 14$. This result immediately implies an algorithm for constructing a planar geometric spanner of a Euclidean graph with stretch factor $\rho \cdot C_{del}$ and degree bounded by k, for any integer parameter $k \geq 14$. Moreover, the resulting spanner contains a Euclidean Minimum Spanning Tree (EMST) as a subgraph. Our second contribution lies in developing the structural results necessary to transfer our analysis and algorithm from Euclidean graphs to unit disk graphs, the usual model for wireless ad-hoc networks. We obtain a very simple distributed, *strictly-localized* algorithm that, given a unit disk graph embedded in the plane, constructs a geometric spanner with the above stretch factor and degree bound, and also containing an EMST as a subgraph. The obtained results dramatically improve the previous results in all aspects, as shown in the paper.

Introduction

Given a set of points P in the plane, the Euclidean graph E on P is defined to be the complete graph whose vertex-set is P. Each edge AB connecting points A and B is assumed to be embedded in the plane as the straight line segment AB; we define its cost to be the Euclidean distance $|AB|$. We define the unit disk graph U to be the subgraph of E consisting of all edges AB with $|AB| \leq 1$.

Let G be a subgraph of E. The cost of a simple path $A = M_0, M_1, ..., M_r = B$ in G is $\sum_{j=0}^{r-1} |M_j M_{j+1}|$. Among all paths between A and B in G, a path with the smallest cost is defined to be a *smallest cost path* and we denote its cost as $c_G(A, B)$. A spanning subgraph H of G is said to be a *geometric spanner* of G if there is a constant ρ such that for every

1998 ACM Subject Classification: C.1.4, F.2.2, G.2.2.

Key words and phrases: Geometric spanner, euclidean graph, unit disk graph, wireless ad-hoc networks.
The work of the first author was supported in part by a DePaul University Competitive Research Grant.

two points $A, B \in G$ we have: $c_H(A,B) \leq \rho \cdot c_G(A,B)$. The constant ρ is called the *stretch factor* of H (with respect to the underlying graph G).

The problem of constructing geometric spanners of Euclidean graphs has recently received a lot of attention due to its applications in computational geometry, wireless computing, and computer graphics (see, for example, the recent book [13] for a survey on geometric spanners and their applications in networks). Dobkin et al. [9] showed that the Delaunay graph is a planar geometric spanner of the Euclidean graph with stretch factor $(1+\sqrt{5})\pi/2 \approx 5.08$. This ratio was improved by Keil et al [10] to $C_{del} = 2\pi/(3\cos(\pi/6)) \approx 2.42$, which currently stands as the best upper bound on the stretch factor of the Delaunay graph. Many researchers believe, however, that the lower bound of $\pi/2$ shown in [7] is also an upper bound on the stretch factor of the Delaunay graph. While Delaunay graphs are good planar geometric spanners of Euclidean graphs, they may have unbounded degree.

Other geometric (sparse) spanners were also proposed in the literature including the Yao graphs [16], the Θ-graphs [10], and many others (see [13]). However, most of these proposed spanners either do not guarantee planarity, or do not guarantee bounded degree.

Bose et al. [2, 3] were the first to show how to extract a subgraph of the Delaunay graph that is a planar geometric spanner of the Euclidean graph with stretch factor ≈ 10.02 and degree bounded by 27. In the context of unit disk graphs, Li et al. [11, 12] gave a distributed algorithm that constructs a planar geometric spanner of a unit disk graph with stretch factor C_{del}; however, the spanner constructed can have unbounded degree. Wang and Li [14, 15] then showed how to construct a bounded-degree planar spanner of a unit disk graph with stretch factor $max\{\pi/2, 1+\pi\sin(\alpha/2)\}\cdot C_{del}$ and degree bounded by $19+2\pi/\alpha$, where $0 < \alpha < 2\pi/3$ is a parameter. Very recently, Bose et. al [5] improved the earlier result in [2, 3] and showed how to construct a subgraph of the Delaunay graph that is a geometric spanner of the Euclidean graph with stretch factor: $max\{\pi/2, 1+\pi\sin(\alpha/2)\}\cdot C_{del}$ if $\alpha < \pi/2$ and $(1+2\sqrt{3}+3\pi/2+\pi\sin(\pi/12))\cdot C_{del}$ when $\pi/2 \leq \alpha \leq 2\pi/3$, and whose degree is bounded by $14 + 2\pi/\alpha$. Bose et al. then applied their construction to obtain a planar geometric spanner of a unit disk graph with stretch factor $max\{\pi/2, 1+\pi\sin(\alpha/2)\}\cdot C_{del}$ and degree bounded by $14 + 2\pi/\alpha$, for any $0 < \alpha \leq \pi/3$. This was the best bound on the stretch factor and the degree.

We have two new results in this paper. We develop structural results about Delaunay graphs that allow us to present a very simple linear-time algorithm that, given a Delaunay graph, constructs a subgraph of the Delaunay graph with stretch factor $1+2\pi(k\cos(\pi/k))^{-1}$ (with respect to the Delaunay graph) and degree at most k, for any integer parameter $k \geq 14$. This result immediately implies an $O(n \lg n)$ algorithm for constructing a planar geometric spanner of a Euclidean graph with stretch factor of $(1+2\pi(k\cos(\pi/k))^{-1})\cdot C_{del}$ and degree at most k, for any integer parameter $k \geq 14$ (n is the number of vertices in the graph). We then translate our work to unit disk graphs and present our second result: a very simple and *strictly-localized* distributed algorithm that, given a unit-disk graph embedded in the plane, constructs a planar geometric spanner of the unit disk graph with stretch factor $(1+2\pi(k\cos(\pi/k))^{-1})\cdot C_{del}$ and degree bounded by k, for any integer parameter $k \geq 14$. This efficient distributed algorithm exchanges no more than $O(n)$ messages in total, and runs in $O(\Delta \lg \Delta)$ local time at a node of degree Δ. We show that both spanners include a Euclidean Minimum Spanning Tree as a subgraph.

Both algorithms significantly improve previous results (described above) in terms of the stretch factor and the degree bound. To show this, we compare our results with previous results in more detail. For a degree bound $k = 14$, our result on Euclidean graphs imply

a bound of at most 3.54 on the stretch factor. As the degree bound k approaches ∞, our bound on the stretch factor approaches $C_{del} \approx 2.42$. The very recent results of Bose et al. [5] achieve a lowest degree bound of 17, and that corresponds to a bound on the stretch factor of at least 23. If Bose et al. [5] allow the degree bound to be arbitrarily large (i.e., to approach ∞), their bound on the stretch factor approaches $(\pi/2) \cdot C_{del} > 3.75$. Our stretch factor and degree bounds for unit disk graphs are the same as our results for Euclidean graphs. The smallest degree bound derived by Bose et al. [5] is 20, and that corresponds to a stretch factor of at least 6.19. If Bose et al. [5] allow the degree bound to be arbitrarily large, then their bound on the stretch factor approaches $(\pi/2) \cdot C_{del} > 3.75$. On the other hand, the smallest degree bound derived in Wang et al. [14, 15] is 25, and that corresponds to a bound of 6.19 on the stretch factor. If Wang et al. [14, 15] allow the degree bound to be arbitrarily large, then their bound on the stretch factor approaches $(\pi/2) \cdot C_{del} > 3.75$. Therefore, even the worst bound of at most 3.54 on the stretch factor corresponding to our lowest bound on the degree $k = 14$, beats the best bound on the stretch factor of at least 3.75 corresponding to arbitrarily large degree in both Bose et al. [5] and Wang et al. [14, 15]!

1. Definitions and Background

We start with the following well known observation:

Observation 1.1. A subgraph H of graph G has stretch factor ρ if and only if for every edge $XY \in G$: the length of a shortest path in H from X to Y is at most $\rho \cdot |XY|$.

For three non-collinear points X, Y, Z in the plane we denote by $\bigcirc XYZ$ the circumscribed circle of triangle $\triangle XYZ$. A *Delaunay triangulation* of a set of points P in the plane is a triangulation of P in which the circumscribed circle of every triangle contains no point of P in its interior. It is well known that if the points in P are *in general position* (i.e., no four points in P are cocircular) then the Delaunay triangulation of P is unique [8]. In this paper—as in most papers in the literature—we shall assume that the points in P are in general position; otherwise, the input can be slightly perturbed so that this condition is satisfied. The *Delaunay graph* of P is defined as the plane graph whose point-set is P and whose edges are the edges of the Delaunay triangulation of P. An alternative definition that we end up using is:

Definition 1.2. An edge XY is in the Delaunay graph of P if and only if there exists a circle through points X and Y whose interior contains no point in P.

It is well known that the Delaunay graph of a set of points P is a spanning subgraph of the Euclidean graph defined on P (i.e., the complete graph on point-set P) whose stretch factor is bounded by $C_{del} = 4\sqrt{3}\pi/9 \approx 2.42$ [10].

Given integer parameter $k > 6$, the *Yao subgraph* [16] of a plane graph G is constructed by performing the following *Yao step* at every point M of G: place k equally-separated rays out of M (arbitrarily defined), thus creating k closed cones of size $2\pi/k$ each, and choose the shortest edge in G out of M (if any) in each cone. The Yao subraph consists of edges in G chosen by *either* endpoint. Note that the degree of a point in the Yao subgraph of G may be unbounded.

Two edges MX, MY incident on a point M in a graph G are said to be *consecutive* if one of the angular sectors determined by MX and MY contains no neighbors of M.

2. Bounded Degree Spanners of Delaunay Graphs

Let P be a set of points in the plane and let E be the complete, Euclidean graph defined on point-set P. Let G be the Delaunay graph of P. This section is devoted to proving the following theorem:

Theorem 2.1. *For every integer $k \geq 14$, there exists a subgraph G' of G such that G' has maximum degree k and stretch factor $1 + 2\pi(k \cos \frac{\pi}{k})^{-1}$.*

A linear time algorithm that computes G' from G is the key component of our proof. This very simple algorithm essentially performs a *modified Yao step* (see Section 2.3) and selects up to k edges out of every point of G. G' is simply the spanning subgraph of G consisting of edges chosen by *both* endpoints.

In order to describe the modified Yao step, we must first develop a better understanding of the structure of the Delaunay graph G. Let CA and CB be edges incident on point C in G such that $\angle BCA \leq 2\pi/k$ and CA is the shortest edge within the angular sector $\angle BCA$. We will show how the above theorem easily follows if, for every such pair of edges CA and CB:

1. we show that there exists a path p from A to B in G of length $|p|$, such that: $|CA| + |p| \leq (1 + 2\pi(k \cos \frac{\pi}{k})^{-1})|CB|$, and
2. we modify the standard Yao step to include the edges of this path in G', in addition to including the edges picked by the standard Yao step but without increasing the number of edges chosen at each point beyond k.

This will ensure that: for any edge $CB \in G$ that is not included in G' by the modified Yao step, there is a path from C to B in G', whose edges are all included in G' by the modified Yao step, and whose cost is at most $(1+2\pi(k \cos \frac{\pi}{k})^{-1})|CB|$. In the lemma below, we prove the existence of this path and show some properties satisfied by edges of this path; we will then modify the standard Yao step to include edges satisfying these properties.

Lemma 2.2. *Let $k \geq 14$ be an integer, and let CA and CB be edges in G such that $\angle BCA \leq 2\pi/k$ and CA is the shortest edge in the angular sector $\angle BCA$. There exists a path $p : A = M_0, M_1, ..., M_r = B$ in G such that:*

(i) $|CA| + \sum_{i=0}^{r-1} |M_i M_{i+1}| \leq (1 + 2\pi(k \cos \frac{\pi}{k})^{-1})|CB|$.
(ii) *There is no edge in G between any pair M_i and M_j lying in the closed region delimited by CA, CB and the edges of p, for any i and j satisfying $0 \leq i < j - 1 \leq r$.*
(iii) $\angle M_{i-1} M_i M_{i+1} > (\frac{k-2}{k})\pi$, *for $i = 1, \cdots, r-1$.*
(iv) $\angle CAM_1 \geq \frac{\pi}{2} - \frac{\pi}{k}$.

We break down the proof of the above lemma into two cases: when $\triangle ABC$ contains no point of G in its interior, and when there are points of G inside $\triangle ABC$. We define some additional notation and terminology first. We define the circle $(O) = \bigcirc ABC$ with center O, and set $\Theta = \angle BCA$. Note that $\angle AOB = 2\Theta \leq 4\pi/k$. We will use $\overset{\frown}{AB}$ to denote the arc of (O) determined by points A and B and facing $\angle AOB$. We will make use of the following easily verified Delaunay graph property:

Proposition 2.3. *If CA and CB are edges of G then the region inside (O) subtended by chord CA and away from B and the region inside (O) subtended by chord CB and away from A contain no points.*

2.1. The Outward Path

We consider first the case when no points of G are inside $\triangle ABC$. Since both CA and CB are edges in G and by Proposition 2.3, the region of (O) subtended by chord AB closer to C has no points of G in its interior. Keil and Gutwin [10] showed that, in this case, there exists a path between A and B in G inside the region of (O) subtended by chord AB away from C, whose length is bounded by the length of $\widehat{AB}$ (see Lemma 1 in [10]). We find it convenient to use a recursive definition of their path (for more details, we refer the reader to [10]):

1. **Base case:** If $AB \in G$, the path consists of edge AB.
2. **Recursive step:** Otherwise, a point must reside in the region of (O) subtended by chord AB and away from C. Let T be such a point with the property that the region of $\bigcirc ATB$ subtended by chord AB closer to T is empty. We call T an *intermediate point* with respect to the pair of points (A, B). Let (O_1) be the circle passing through A and T whose center O_1 lies on segment AO and let (O_2) be the circle passing through B and T whose center O_2 lies on segment BO. Then both (O_1) and (O_2) lie inside (O), and $\angle AO_1T$ and $\angle TO_2B$ are both less than $\angle AOB \leq 4\pi/k$. Moreover, the region of (O_1) subtended by chord AT that contains O_1 is empty, and the region of (O_2) subtended by chord BT and containing O_2 is empty. Therefore, we can recursively construct a path from A to T and a path from T to B, and then concatenate them to obtain a path from A to B.

Definition 2.4. We call the path constructed above the *outward path* between A and B.

Keil and Gutwin [10], from this point on, use a purely geometric argument (with no use of Delaunay graph properties) to show that the length of the obtained path $A = M_0, M_1, \cdots, M_r = B$ (where each point M_p, for $p = 1, \cdots, r-1$, is an intermediate point with respect to a pair (M_i, M_j), where $0 \leq i < p < j \leq r$) is smaller than the length of $\widehat{AB}$. Figure 1 illustrates an outward path between A and B.

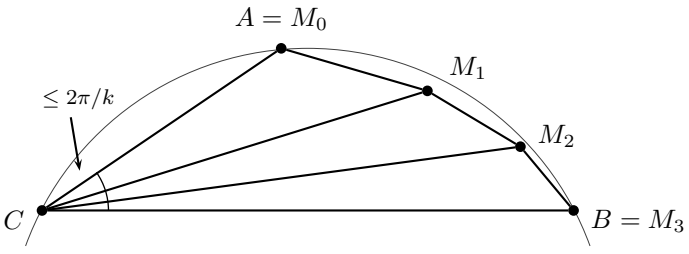


Figure 1: Illustration of an outward path.

Proposition 2.5. *In every recursive step of the outward path construction described above, if M_p is an intermediate point with respect to a pair of points (M_i, M_j), then:*

(a) *there is a circle passing through C and M_p that contains no point of G, and*
(b) *circles $\bigcirc CM_iM_p$ and $\bigcirc CM_jM_p$ contain no points of G except, possibly, in the region subtended by chords M_iM_p and M_pM_j, respectively, away from C.*

Proof. We assume, by induction, that there are circles (O_{M_i}) and (O_{M_j}) passing through C and M_i, and C and M_j, respectively, containing no points of G, and that the circle $(O) = \bigcirc CM_iM_j$ contains no point of G in the interior of the region R' subtended by chord M_iM_j closer to C. (This is certainly true in the base case because $CA, CB \in G$, by Proposition 2.3, and by our initial assumptions).

Since M_iM_j is not an edge in G, the point M_p chosen in the construction is the point with the property that the region R of $\bigcirc M_iM_pM_j$ subtended by chord M_iM_j away from C, contains no point of G. Then the circle passing through C and M_p and tangent to $\bigcirc M_iM_pM_j$ at M_p is completely inside $(O_{M_i}) \cup (O_{M_j}) \cup R \cup R'$, and therefore devoid of points of G. This proves part (a).

The region of $\bigcirc CM_iM_p$ subtended by chord M_iM_p and containing C is inside $(O_{M_i}) \cup R \cup R'$, and therefore contains no point of G in its interior. The same is true for the region of $\bigcirc CM_jM_p$ subtended by chord M_jM_p and containing C, and part (b) holds as well. ∎

We are now ready to prove Lemma 2.2 in the case when no point of G lies inside $\triangle ABC$. In this case we define the path in Lemma 2.2 to be the outward path between A and B.

Proof of Lemma 2.2 for the case of outward path.

(*i*) With $\Theta = \angle BCA$, we have $\mid \overset{\frown}{AB} \mid = 2\Theta \cdot |OA|$ and $\sin\Theta = |AB|/(2|OA|)$. We note that $|CA| + \mid \overset{\frown}{AB} \mid$ is largest when $|CA| = |CB|$, i.e. when CA and CB are symmetrical with respect to the diameter of $\bigcirc CAB$ passing through C; this follows from the fact that the perimeter of a convex body is not smaller than the perimeter of a convex body containing it (see page 42 in [1]). If $|CA| = |CB|$, $\sin\frac{\Theta}{2} = \frac{|AB|}{2|CB|}$. Using elementary trigonometry, it follows from the above facts and from $|CA| \leq |CB|$ that:

$$\begin{aligned} |CA| + \mid \overset{\frown}{AB} \mid &\leq |CB| + 2\Theta \cdot |OA| = |CB| + (\frac{\Theta}{\sin\Theta}) \cdot |AB| = |CB| + (\frac{\Theta}{\cos\frac{\Theta}{2}}) \cdot |CB| \\ &\leq (1 + 2\pi(k\cos\frac{\pi}{k})^{-1})|CB|. \end{aligned}$$

The last inequality follows from $\Theta \leq 2\pi/k$ and $k > 2$.

(*ii*) If M_iM_j was an edge in G then, for every p between i and j, the circle $\bigcirc M_iM_pM_j$ would not contain C. This, however, contradicts part (a) of Proposition 2.5.

(*iii*) If the outward path contains a single intermediate point M_1, then since M_1 lies inside $(O) = \bigcirc CAB$, $\angle AM_1B \geq \pi - \angle AOB/2 \geq \pi - 2\pi/k = (k-2)\pi/k$ (note that $\angle AOB = 2 \cdot \angle ACB$), as desired. Now the statement follows by induction on the number of steps taken to construct the outward path between A and B, using the fact (proved in [10]) that each angle $\angle M_{i-1}O_iM_{i+1}$ at the center of the circle (O_i) defining the intermediate point M_i, is bounded by $\angle AOB$.

(*iv*) This follows from the fact that $\angle CAM_1 \geq \angle CAB \geq \pi/2 - \pi/k$. The last inequality is true because $|CA| \leq |CB|$ and $\angle BCA \leq 2\pi/k$ in $\triangle CAB$.

∎

2.2. The Inward Path

We consider now the case when the interior of $\triangle ABC$ contains points of G. Let S be the set of points consisting of points A and B plus all the points interior to $\triangle ABC$ (note

that $C \notin S$). Let $CH(S)$ be the points on the convex hull of S. Then $CH(S)$ consists of points $N_0 = A$ and $N_s = B$, and points $N_1, ..., N_{s-1}$ of G interior to $\triangle ABC$. We have the following proposition:

Proposition 2.6. *For every $i = 1, \cdots, s-1$:*

(a) $CN_i \in G$,
(b) $|CN_i| \leq |CN_{i+1}|$, *and*
(c) $\angle N_{i-1}N_iN_{i+1} \geq \pi$, *where $\angle N_{i-1}N_iN_{i+1}$ is the angle facing point C.*

Proof. These follow follow from the following facts: CA and CB are edges in G, CA is the shortest edge in its cone, and hence $|CA| \leq |CN_i|$, for $i = 0, \cdots, s$, and points $N_0, \cdots, N_s$ are on $CH(S)$ in the listed order. ∎

Since $|CN_i| \leq |CN_{i+1}|$ and no point of G lies inside $\triangle N_iCN_{i+1}$ (N_i and N_{i+1} are on $CH(S)$), CN_i is the shortest edge in the angular sector $\angle N_iCN_{i+1}$. Since $\angle N_iCN_{i+1} \leq \angle BCA \leq 2\pi/k$, by Lemma 2.2 there exists an outward path P_i between N_i and N_{i+1}, for every $i = 0, 1, \cdots, s-1$, satisfying all the properties of Lemma 2.2. Let $A = M_0, M_1, \cdots, M_r = B$ be the concatenation of the paths P_i, for $i = 0, \cdots, r-1$.

Definition 2.7. We call the path $A = M_0, M_1, \cdots, M_r = B$ constructed above the *inward path* between A and B.

Figure 2 illustrates an inward path between A and B.

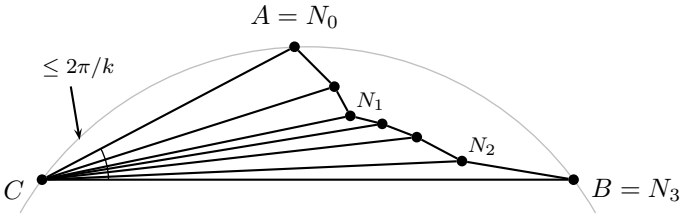


Figure 2: Illustration of an inward path.

We now prove Lemma 2.2 in the case when there are points of G interior to $\triangle ABC$. In this case we define the path in Lemma 2.2 to be the inward path between A and B.

Proof of Lemma 2.2 for the case of inward path.

(*i*) Define A'' to be a point on the half-line $[CA$ such that $|CA''| = |CB|$, and let $(O'') = \bigcirc CA''B$. Denote by α'' the length of the arc of $\bigcirc CA''B$ subtended by chord $A''B$ and facing $\angle A''CB$. For every $i = 0, 1, \cdots, s-1$, we define arc α_i to be the arc of $\bigcirc CN_iN_{i+1}$ subtended by chord N_iN_{i+1} and facing $\angle N_iCN_{i+1}$. For every $i = 0, 1, ..., s-1$, we define N_i' to be the point on the half-line $[CN_i$ such that $|CN_i'| = |CN_{i+1}|$, (O_i) to be the circle $\bigcirc CN_i'N_{i+1}$, and α_i' to be the arc of (O_i) subtended by chord $N_i'N_{i+1}$ and facing $\angle N_i'CN_{i+1}$. Finally, for every $i = 0, \cdots, s-1$, we define N_i'' to be the point of intersection of the half-line $[CN_i$ and circle (O''), and α_i'' to be the arc of (O'') subtended by chord $N_i''N_{i+1}''$ and facing $\angle N_i''CN_{i+1}''$. As shown in section 2.1, the length of the outward path P_i between N_i and N_{i+1} is bounded by the length of α_i. Since the convex body C_1 delimited by CN_i, CN_{i+1} and α_i is contained inside the convex body C_2 delimited by CN_i', CN_{i+1} and α_i',

by [1], the perimeter of C_1 is not larger than that of C_2. Denoting by $|P_i|$ the length of path P_i, we get:

$$|P_i| \leq |N_iN_i'| + \alpha_i', \quad i = 1, \cdots, s-1. \tag{2.1}$$

Since (O_i) and (O'') are concentric circles (of center C), and the radius of (O_i) is not larger than that of (O''), we have $\alpha_i' \leq \alpha_i''$, for $i = 0, \cdots, s-1$. It follows from Inequality (2.1) that:

$$|P_i| \leq |N_iN_i'| + \alpha_i'', \quad i = 1, \cdots, s-1. \tag{2.2}$$

Using Inequalities (2.1) and (2.2) we get:

$$|CA| + \sum_{i=0}^{s-1} |P_i| \leq |CA| + \sum_{i=0}^{s-1} |N_iN_i'| + \sum_{i=0}^{s-1} \alpha_i''. \tag{2.3}$$

Noting that $\sum_{i=0}^{s-1} |N_iN_i'| = |CB| - |CA|$, that $\sum_{i=0}^{r-1} \alpha_i'' = \alpha''$, and using the same argument as in part (i) of Lemma 2.2) completes the proof.

(ii) Since $CN_p \in G$ for $p = 1, \cdots, s-1$ by part (a) of Proposition 2.6, by planarity of G, if such an edge between two points M_i and M_j exists, then M_i and M_j must belong to an outward path between two points N_p and N_{p+1} of $CH(S)$. But this contradicts part (ii) of Lemma 2.2 for the case of the outward path applied to N_p and N_{p+1}.

(iii) For each $i = 0, \cdots, r$, either $M_i = N_j \in CH(S)$, or M_i is an intermediate point on the outward path between two points N_p and N_q in $CH(S)$. In the former case $\angle M_{i-1}M_iM_{i+1} \geq \angle N_{j-1}M_iN_{j+1} \geq \pi \geq (k-2)\pi/k$ for $k \geq 14$ (N_{j-1} and N_j are points before and after $M_i = N_j$ on $CH(S)$), by part (c) of Proposition 2.6. In the latter case $\angle M_{i-1}M_iM_{i+1} \geq (k-2)\pi/k$ by the proof of part (iii) of Lemma 2.2 applied to the outward path between N_p and N_q.

(iv) This follows from $|CA| = |CM_0| \leq |CM_1|$ and $\angle ACM_1 \leq \angle ACB \leq 2\pi/k$, in triangle $\triangle CAM_1$.

■

2.3. The Modified Yao Step

We now augment the *Yao step* so edges forming the paths described in Lemma 2.2 are included in G', in addition to the edges chosen in the standard Yao step. Lemma 2.2 says that consecutive edges on such paths form moderately large angles. The modified Yao step will ensure that consecutive edges forming large angles are included in G'. The algorithm is described in Figure 3.

Since the algorithm selects at most k edges incident on any point M and since only edges chosen by both endpoints are included in G', each point has degree at most k in G'.

Before we complete the proof of Theorem 2.1, we show that the running time of the algorithm is linear. Note first that all edges incident on point M of degree Δ can be mapped to the k cones around M in linear time in Δ. Then, the shortest edge in every cone can be found in time $O(\Delta)$ (step 2. in the algorithm). Since k is a constant, selecting the $\ell/2$ edges clockwise (or counterclockwise) from a sequence of $ell < k$ empty cones around M (step 3.1.) can be done in $O(\Delta)$ time. Noting that the total number of edges in G is linear in the number of vertices completes the analysis.

To complete the proof of Theorem 2.1, all we need to do is show:

Algorithm **Modified Yao step**

INPUT: *A Delaunay graph G; integer $k \geq 14$*
OUTPUT: *A subgraph G' of G of maximum degree k*

1. define k disjoint cones of size $2\pi/k$ around every point M in G;
2. in every non-empty cone, select the shortest edge incident on M in this cone;
3. **for** every maximal sequence of $\ell \geq 1$ consecutive empty cones:
   3.1. **if** $\ell > 1$ **then** select the first $\lfloor \ell/2 \rfloor$ unselected incident edges on M clockwise from the sequence of empty cones and the first $\lceil \ell/2 \rceil$ unselected edges incident on M counterclockwise from the sequence of empty cones;
   3.2. **else** (i.e., $\ell = 1$) let MX and MY be the incident edges on M clockwise and counterclockwise, respectively, from the empty cone; **if** either MX or MY is selected **then** select the other edge (in case it has not been selected); **otherwise** select the shorter edge between MX and MY breaking ties arbitrarily;
4. G' is the spanning subgraph of G consisting of edges selected by both endpoints.

Figure 3: The modified Yao Step.

Lemma 2.8. *If edge CB is not selected by the algorithm, let CA be the shortest edge in the cone out of C to which CB belongs. Then the edges of the path described in Lemma 2.2 are included in G' by the algorithm.*

Proof. For brevity, instead of saying that the algorithm **Modified Yao Step** selects an edge MX out of a point M, we will say that M selects edge MX. To get started, it is obvious that C will select edge CA.

By part (iv) of Lemma 2.2, the angle $\angle CAM_1 \geq \pi/2 - \pi/k \geq 6\pi/k$ for $k \geq 14$. Therefore, at least two empty cones must fall within the sector $\angle CAM_1$ determined by the two consecutive edges CA and AM_1, and edges AC and AM_1 will both be selected by A. Since edge CA is also selected by point C, edge $AC \in G'$.

By part (iii) of Lemma 2.2, for every $i = 1, 2, \cdots, r-1$, the angle $\angle M_{i-1}M_iM_{i+1} \geq (k-2)\pi/k \geq 10\pi/k$ for $k \geq 12$, and hence at least four cones fall within the angular sector $\angle M_{i-1}M_iM_{i+1}$. Since by part *(ii)* of Lemma 2.2 M_iC is the only possible edge inside the angular sector $\angle M_{i-1}M_iM_{i+1}$, it is easy to see that regardless of the position of these four cones with respect to edge M_iC, M_i ends up selecting all edges M_iM_{i-1}, M_iM_{i+1} and M_iC in steps 2 and/or 3 of the algorithm. Since we showed above that A selects edge AM_1, this shows that all edges M_iM_{i+1}, for $i = 0, \cdots, r-2$, are selected by both their endpoints, and hence must be in G'. Moreover, edge $M_{r-1}M_r = M_{r-1}B$ is selected by point M_{r-1}.

We now argue that edge BM_{r-1} will be selected by B. First, observe that $|BM_{r-1}| \leq |\overset{\frown}{AB}| < |CB|$. Let CD be the other consecutive edge to CB in G (other than CM_{r-1}). Because C does not select B, it follows that $\angle M_{r-1}CD \leq 6\pi/k$. Otherwise, since CM_{r-1} and CB are in the same cone, two empty cones would fall within the sector $\angle BCD$ and C would select B. Since CB is an edge in G, by the characterization of Delaunay edges [8], $\angle CM_{r-1}B + \angle CDB \leq \pi$. By considering the quadrilateral $CDBM_{r-1}$, we have $\angle M_{r-1}CD + \angle DBM_{r-1} \geq \pi$. This, together with the fact that $\angle M_{r-1}CD \leq 6\pi/k$, imply that $\angle DBM_{r-1} \geq (k-6)\pi/k \geq 8\pi/k$, for $k \geq 14$. Therefore, $\angle DBM_{r-1}$ contains at least

three cones of size $2\pi/k$ out of B. If one of these cones falls within the angular sector $\angle CBM_{r-1}$ then, since $|M_{r-1}B| < |CB|$, BM_{r-1} must have been selected out of B.

Suppose now that $\angle CBM_{r-1}$ contains no cone inside and hence $\angle CBM_{r-1} < 4\pi/k$. If one of these three cones within sector $\angle DBM_{r-1}$ contains edge CB, then the remaining two cones must fall within $\angle DBC$ and BM_{r-1} will get selected out of B when considering the sequence of at least two empty cones contained within $\angle CBD$. Suppose now that all three empty cones fall within $\angle CBD$. Then we have $\angle CBD \geq 6\pi/k$.

If $\angle M_{r-1}CD \geq 4\pi/k$, then since $M_{r-1}C$ and CB belong to the same cone, the sector $\angle BCD$ must contain an empty cone. Because D is exterior to $\bigcirc CBM_{r-1}$, $\angle CBM_{r-1} < 4\pi/k$, and $\angle M_{r-1}CB \leq 2\pi/k$, it follows that $\angle CDB < \angle M_{r-1}CB + \angle CBM_{r-1} < 6\pi/k < \angle DBC$. Therefore, by considering the triangle $\triangle CDB$, we note that $|CB| < |CD|$. But then edge CB would have been selected by C in step 3 since the sector $\angle BCD$ contains an empty cone, a contradiction.

It follows that $\angle M_{r-1}CD \leq 4\pi/k$, and therefore $\angle M_{r-1}BD \geq (k-4)\pi/k \geq 10\pi/k$ for $k \geq 14$. This means that at least four cones are contained inside sector $\angle DBM_{r-1}$. It is easy to check now that regardless of the placement of the edge BC with respect to these cones, edge BM_{r-1} is always selected out of B by the algorithm. This completes the proof. ∎

Corollary 2.9. *A Euclidean Minimum Spanning Tree (EMST) on P is a subgraph of G'.*

Proof. It is well known that a Delaunay graph (G) contains a EMST. If an edge CB is not in G', then, by Lemma 2.8, a path from C to B is included in G'. All edges on this path are no longer than CB, so there is a EMST not including CB. ∎

Since a Delaunay graph of a Euclidean graph of n points can be computed in time $O(n \lg n)$ [8] and has stretch factor $C_{del} \approx 2.42$, we have the following theorem.

Theorem 2.10. *There exists an algorithm that, given a set P of n points in the plane, computes a plane geometric spanner of the Euclidean graph on P that contains a EMST, has maximum degree k, and has stretch factor $(1 + 2\pi(k \cos \frac{\pi}{k})^{-1}) \cdot C_{del}$, where $k \geq 14$ is an integer. Moreover, the algorithm runs in time $O(n \lg n)$.*

3. Geometric Spanners of Unit Disk Graphs

In this section we generalize our planar geometric spanner algorithm to unit disk graphs. Unit disk graphs model wireless ad-hoc and sensor networks and, for packet routing and other applications, a bounded-degree planar geometric spanner of the wireless network is often desired. Due to the limited computational power of the network devices and the requirement that the network be robust with respect to device joining and leaving the network, the construction/algorithm should ideally be *strictly-localized*: the computation performed at a point depends solely on the information available at the point and its d-hop neighbors, for some constant d (in our case $d = 2$). In particular, no global propagation of information should take place in the network.

The results in the previous section do not carry over to unit disk graphs because not all Delaunay graph edges on a point-set P are unit disk edges. However, if U is the unit disk graph on points in P and $UDel(U)$ is the subgraph of the Delaunay graph on P obtained by deleting edges of length greater than one unit, then $UDel(U)$ is a connected, planar, spanning subgraph of U with stretch factor bounded by C_{del} (see [11, 4]). Therefore, if we

apply the results from the previous section to $UDel(U)$ and observe that all edges on the path defined in Lemma 2.2 must be unit disk edges (given that edges CA and CB are), it is easy to see that Theorem 2.1 and Theorem 2.10 carry over to unit disk graphs. The only problem, however, is that the construction of $UDel(U)$ cannot be done in a strictly-localized manner.

To solve this problem, Wang et al. [11, 12] introduced a subgraph of U denoted $LDel^{(2)}(U)$. It was shown in [11, 12] that $LDel^{(2)}(U)$ is a planar supergraph of $UDel(U)$, and hence also has stretch factor bounded by C_{del}. Moreover, the results in [6, 15] show how $LDel^{(2)}(U)$ can be computed with a strictly-localized distributed algorithm exchanging no more than $O(n)$ messages in total (n is the number of points in U), and having a local processing time of $O(\Delta \lg \Delta) = O(n \lg n)$ at a point of degree Δ. In a style similar to Definition 1.2, $LDel^{(2)}(U)$ can be defined as follows:

Definition 3.1. An edge XY of U is in $LDel^{(2)}(U)$ if and only if there exists a circle through points X and Y whose interior contains no point of U that is a 2-hop neighbor of X or Y.

We will use $G = LDel^{(2)}(U)$ as the underlying subgraph of U to replace the Delaunay graph G used in the previous section. We note that G is planar, is a supergraph of $UDel(U)$, and hence has stretch factor C_{del}. To translate our results to unit disk graphs, we need to show that the inward and outward paths are still well defined in G. In particular, we need to show that Lemma 2.2 holds for $G = LDel^{(2)}(U)$. We outline the general approach and omit the details for lack of space.

The following is equivalent to Proposition 2.3:

Lemma 3.2. *If CA and CB are edges of G then the region of $(O) = \bigcirc ABC$ subtended by chord CA and away from B and the region of (O) subtended by chord CB and away from A contain no points that are two hop neighbors of A, B and C.*

Proof. By symmetry it is enough to prove the lemma for the region of (O) subtended by chord CA and away from B. By Definition 3.1, there is a circle (O_{CA}) passing through C and A whose interior is empty of any point within two hops of C or A. The region of (O) subtended by chord CA and away from B is inside this circle, so we only need to argue that it doesn't contain two hop neighbors of B either. If it did, say point X, then any neighbor of X and B would have to be a neighbor of C or A as well, a contradiction. ∎

With this lemma in hand, the recursive construction of the outward path given in Subsection 2.1 can be applied to the graph $G = LDel^{(2)}(U)$. The following proposition for $G = LDel^{(2)}(U)$ corresponds to Proposition 2.5 for Delaunay graphs and is proven in an equivalent manner:

Proposition 3.3. *In every recursive step of the outward path construction, if M_p is an intermediate point with respect to a pair of points (M_i, M_j), then:*

(a) *there is a circle passing through C and M_p that contains no point of G that is a two-hop neighbor of C or M_p, and*
(b) *circles $\bigcirc CM_iM_p$ and $\bigcirc CM_jM_p$ contain no points of G that are two-hop neighbors of C, M_i and M_p and C, M_j, and M_p, respectively, except, possibly, in the region subtended by chords M_iM_p and M_pM_j, respectively, away from C.*

With this proposition, we can show that Lemma 2.2 holds true for $G = LDel^{(2)}(U)$ for outward paths. It holds for inward paths as well, using the same argument as in Section 2.2.

Finally, it is obvious how the **Modified Yao Step** algorithm in Section 2.3 can be easily described as a strictly-localized algorithm. We can show, therefore, the following theorem:

Theorem 3.4. *There exists a distributed strictly-localized algorithm that, given a set P of n points in the plane, computes a plane geometric spanner of the unit disk graph on P that contains a EMST, has maximum degree k, and has stretch factor $(1+2\pi(k\cos\frac{\pi}{k})^{-1})\cdot C_{del}$, for any integer $k \geq 14$. Moreover, the algorithm exchanges no more than $O(n)$ messages in total, and has a local processing time of $\Delta \lg \Delta$ at a point of degree Δ.*

Due to the strictly-localized nature of the algorithm, the algorithm is very robust to topological changes (such as wireless devices moving or joining or leaving the network), an essential property for the application of the algorithm in a wireless ad-hoc environment.

References

[1] R. Benson. *Euclidean Geometry and Convexity*. Mc-Graw Hill, New York, 1966.

[2] P. Bose, J. Gudmundsson, and M. Smid. Constructing plane spanners of bounded degree and low weight. In *Proceedings of the 10th Annual European Symposium on Algorithms*, volume 2461 of *Lecture Notes in Computer Science*, pages 234–246. Springer, 2002.

[3] P. Bose, J. Gudmundsson, and M. Smid. Constructing plane spanners of bounded degree and low weight. *Algorithmica*, 42(3-4):249–264, 2005.

[4] P. Bose, A. Maheshwari, G. Narasimhan, M. Smid, and N. Zeh. Approximating geometric bottleneck shortest paths. *Computational Geometry: Theory and Applications*, 29:233–249, 2004.

[5] P. Bose, M. Smid, and D. Xu. Diamond triangulations contain spanners of bounded degree. *To appear in Algorithmica*, 2007.

[6] G. Calinescu. Computing 2-hop neighborhoods in Ad Hoc wireless networks. In *Proceedingsof the 2nd International Conference on Ad-Hoc, Mobile, and Wireless Networks*, volume 2865 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 2003.

[7] P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computers and System Sciences.*, 39(2):205–219, 1989.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[9] D. Dobkin, S. Friedman, and K. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete Computational Geometry*, 5(4):399–407, 1990.

[10] J. Keil and C. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.

[11] X.-Y. Li, G. Calinescu, and P.-J. Wan. Distributed construction of planar spanner and routing for ad hoc wireless networks. In *Proceedings of the IEEE INFOCOM*, 2002.

[12] X.-Y. Li, G. Calinescu, P.-J. Wan, and Y. Wang. Localized delaunay triangulation with application in Ad Hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems.*, 14(10):1035–1047, 2003.

[13] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[14] Y. Wang and X.-Y. Li. Localized construction of bounded degree and planar spanner for wireless ad hoc networks. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing*, pages 59–68. ACM, 2003.

[15] Y. Wang and X.-Y. Li. Localized construction of bounded degree and planar spanner for wireless ad hoc networks. *MONET*, 11(2):161–175, 2006.

[16] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 421-432
www.stacs-conf.org

THE FROBENIUS PROBLEM IN A FREE MONOID

JUI-YI KAO [1], JEFFREY SHALLIT [1], AND ZHI XU [1]

[1] School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, CANADA
E-mail address: JY.academic@gmail.com
E-mail address, Jeffrey Shallit: shallit@graceland.uwaterloo.ca
E-mail address, Zhi Xu: z5xu@cs.uwaterloo.ca

Abstract. The classical Frobenius problem over $\mathbb{N}$ is to compute the largest integer g not representable as a non-negative integer linear combination of non-negative integers $x_1, x_2, \ldots, x_k$, where $\gcd(x_1, x_2, \ldots, x_k) = 1$. In this paper we consider novel generalizations of the Frobenius problem to the noncommutative setting of a free monoid. Unlike the commutative case, where the bound on g is quadratic, we are able to show exponential or subexponential behavior for several analogues of g, with the precise bound depending on the particular measure chosen.

1. Introduction

Let $x_1, x_2, \ldots, x_k$ be positive integers. It is well-known that every sufficiently large integer can be written as a non-negative integer linear combination of the x_i if and only if $\gcd(x_1, x_2, \ldots, x_k) = 1$. The famous *Frobenius problem* (so-called because, according to Brauer [2], "Frobenius mentioned it occasionally in his lectures") is the following:

Given positive integers $x_1, x_2, \ldots, x_k$ with $\gcd(x_1, x_2, \ldots, x_k) = 1$, find the largest positive integer $g(x_1, x_2, \ldots, x_k)$ which *cannot* be represented as a non-negative integer linear combination of the x_i.

Although it seems simple at first glance, the Frobenius problem on positive integers has many subtle and intriguing aspects that continue to elicit study. A recent book by Ramírez Alfonsín [23] lists over 400 references on this problem. Applications to many different fields exist: to algebra [19]; the theory of matrices [11], counting points in polytopes [1]; the problem of efficient sorting using Shellsort [17], the theory of Petri nets [25]; the liveness of weighted circuits [8]; etc.

Generally speaking, research on the Frobenius problem can be classified into three different areas:

1998 ACM Subject Classification: F.4.3.
Key words and phrases: combinatorics on words, Frobenius problem, free monoid.
Reseach supported by a grant from NSERC.

- Formulas or algorithms for the exact computation of $g(x_1, \ldots, x_k)$, including formulas for g where the x_i obey certain relations, such as being in arithmetic progression;
- The computational complexity of the problem;
- Good upper or lower bounds on $g(x_1, \ldots, x_k)$.

For $k = 2$, it is folklore that

$$g(x_1, x_2) = x_1 x_2 - x_1 - x_2; \tag{1.1}$$

this formula is often attributed to Sylvester [24], although he did not actually state it. Eq. (1.1) gives an efficient algorithm to compute g for two elements. For $k = 3$, efficient algorithms have been given by Greenberg [15] and Davison [10]; if $x_1 < x_2 < x_3$, these algorithms run in time bounded by a polynomial in $\log x_3$. Kannan [18] gave a very complicated algorithm that runs in polynomial time in $\log x_k$ if k is fixed, but is wildly exponential in k. However, Ramírez Alfonsín [22] proved that the general problem is NP-hard, under Turing reductions, by reducing from the integer knapsack problem. So it seems very likely that there is no simple formula for computing $g(x_1, x_2, \ldots, x_k)$ for arbitrary k. Nevertheless, recent work by Einstein, Lichtblau, Strzebonski, and Wagon [12] shows that in practice the Frobenius number can be computed relatively efficiently, even for very large numbers, at least for $k \leq 8$.

Another active area of interest is estimating how big g is in terms of $x_1, x_2, \ldots, x_k$ for $x_1 < x_2 < \cdots < x_k$. It is known, for example, that $g(x_1, x_2, \ldots, x_k) < x_k^2$. This follows from Wilf's algorithm [26]. Many other bounds are known.

One can also study variations on the Frobenius problem. For example, given positive integers $x_1, x_2, \ldots, x_k$ with $\gcd(x_1, x_2, \ldots, x_k) = 1$, what is the number $f(x_1, x_2, \ldots, x_k)$ of positive integers not representable as a non-negative integer linear combination of the x_i? Sylvester, in an 1884 paper [24], showed that $f(x_1, x_2) = \frac{1}{2}(x_1 - 1)(x_2 - 1)$.

Our goal in this paper is to generalize the Frobenius problem to the setting of a free monoid. In this framework, we start with a finite, nonempty alphabet Σ, and consider the set of all finite words Σ^*. Instead of considering integers $x_1, x_2, \ldots, x_k$, we consider words $x_1, x_2, \ldots, x_k \in \Sigma^*$. Instead of considering linear combinations of integers, we instead consider the languages $\{x_1, x_2, \ldots, x_k\}^*$ and $x_1^* x_2^* \cdots x_k^*$. Actually, we consider several additional generalizations, which vary according to how we measure the size of the input, conditions on the input, and measures of the size of the result. For an application of the noncommutative Frobenius problem, see Clément, Duval, Guaiana, Perrin, and Rindone [9].

In sections 2 and 3, we introduce the definition of the generalized Frobenius problem. In sections 4 and 5, we discuss the state complexity of this generalized problem. In sections 5 and 6, we will discuss the longest length and number of omitted words, respectively.

In order to motivate our definitions, we consider the easiest case first: where $\Sigma = \{\mathtt{0}\}$, a unary alphabet.

2. The unary case

Suppose $x_i = \mathtt{0}^{a_i}$, where $a_i \in \mathbb{N}$ for $1 \leq i \leq k$. The Frobenius problem is evidently linked to many problems over unary languages. It figures, for example, in estimating the size of the smallest DFA equivalent to a given NFA [7].

If $L \subseteq \Sigma^*$, by $\overline{L}$ we mean $\Sigma^* - L$, the complement of L. If L is a finite language, by $|L|$ we mean the cardinality of L. Evidently we have

Proposition 2.1. *Suppose* $x_i = \mathtt{0}^{a_i}$ *where* $a_i \in \mathbb{N}$ *for* $1 \le i \le k$, *and write* $S = \{x_1, x_2, \ldots, x_k\}$. *Then* S^* *is co-finite if and only if* $\gcd(a_1, a_2, \ldots, a_k) = 1$. *Furthermore, if* S^* *is co-finite, then the length of the longest word in* $\overline{S^*}$ *is* $g(a_1, a_2, \ldots, a_k)$, *and* $|\overline{S^*}| = f(a_1, a_2, \ldots, a_k)$.

This result suggests that one appropriate noncommutative generalization of the condition $\gcd(a_1, a_2, \ldots, a_k) = 1$ is that $S^* = \{x_1, x_2, \ldots, x_k\}^*$ be co-finite, and one appropriate generalization of the g function is the length of the longest word not in S^*.

But there are other possible generalizations. Instead of measuring the length of the longest omitted word, we could instead consider the *state complexity* of S^*. By the state complexity of a regular language L, written $\mathrm{sc}(L)$, we mean the number of states in the (unique) minimal deterministic finite automaton (DFA) accepting L. In the unary case, this alternate measure has a nice expression in terms of the ordinary Frobenius function:

Theorem 2.2. *Let* $\gcd(a_1, a_2, \ldots, a_k) = 1$. *Then*

$$\mathrm{sc}(\{\mathtt{0}^{a_1}, \mathtt{0}^{a_2}, \ldots, \mathtt{0}^{a_k}\}^*) = g(a_1, a_2, \ldots, a_k) + 2.$$

Proof. Let $L = \{\mathtt{0}^{a_1}, \mathtt{0}^{a_2}, \ldots, \mathtt{0}^{a_k}\}^*$. Since $\gcd(a_1, a_2, \ldots, a_k) = 1$, every word of length $> g(a_1, a_2, \ldots, a_k)$ is contained in L. Thus we can accept L with a DFA having $g(a_1, \ldots, a_k)+2$ states, using a "tail" of $g(a_1, \ldots, a_k) + 1$ states and a "loop" of one accepting state. Thus $\mathrm{sc}(L) \le g(a_1, a_2, \ldots, a_k) + 2$.

To see $\mathrm{sc}(L) \ge g(a_1, a_2, \ldots, a_k) + 2$, we show that the words $\epsilon, \mathtt{0}, \mathtt{0}^2, \ldots, \mathtt{0}^{g(a_1,\ldots,a_k)+1}$ are pairwise inequivalent under the Myhill-Nerode equivalence relation. Pick $\mathtt{0}^i$ and $\mathtt{0}^j$, $0 \le i < j \le g(a_1, \ldots, a_k) + 1$. Choose $z = \mathtt{0}^{g(a_1,\ldots,a_k)-i}$. Then $\mathtt{0}^i z = \mathtt{0}^{g(a_1,\ldots,a_k)} \notin L$, while $\mathtt{0}^j z = \mathtt{0}^{g(a_1,\ldots,a_k)+j-i} \in L$, since $j > i$. ∎

Corollary 2.3. *Let* $\gcd(a_1, \ldots, a_k) = d$. *Then*

$$\mathrm{sc}(\{\mathtt{0}^{a_1}, \mathtt{0}^{a_2}, \ldots, \mathtt{0}^{a_k}\}^*) = d\Big(g(a_1/d, a_2/d, \ldots, a_k/d) + 1\Big) + 1.$$

Hence it follows that $\mathrm{sc}(\{\mathtt{0}^{a_1}, \mathtt{0}^{a_2}, \ldots, \mathtt{0}^{a_k}\}^*) = O(a^2)$ for $a = \max_{1\le i\le k} a_i$. Furthermore, this bound is essentially optimal; since $g(n, n+1) = n^2 - n - 1$, there exist examples with $\mathrm{sc}(\{\mathtt{0}^{a_1}, \mathtt{0}^{a_2}, \ldots, \mathtt{0}^{a_k}\}^*) = \Omega(a^2)$.

3. The case of larger alphabets

We now turn to the main results of the paper. Given as input a list of words $x_1, x_2, \ldots, x_k$, not necessarily distinct, and defining $S = \{x_1, x_2, \ldots, x_k\}$, we can measure the size of the input in a number of different ways:

(a) k, the number of words;
(b) $n = \max_{1\le i\le k} |x_i|$, the length of the longest word;
(c) $m = \sum_{1\le i\le k} |x_i|$, the total number of symbols;
(d) $\mathrm{sc}(\{x_1, x_2, \ldots, x_k\})$, the state complexity of the language represented by the input.

We may impose various conditions on the input:

(i) Each x_i is defined over the unary alphabet;
(ii) $S^* = \{x_1, x_2, \ldots, x_k\}^*$ is co-finite
(iii) $k = 2$, or k is fixed.

And finally, we can explore various measures on the size of the result:

(1) $\mathcal{L} = \max_{x \in \Sigma^* - S^*} |x|$, the length of the longest word not in S^*;
(2) $\mathcal{K} = \max_{x \in \Sigma^* - x_1^* x_2^* \cdots x_k^*} |x|$, the length of the longest word not in $x_1^* x_2^* \cdots x_k^*$;
(3) $\mathcal{S} = \mathrm{sc}(S^*)$, the state complexity of S^*;
(4) $\mathcal{M} = |\Sigma^* - S^*|$, the number of words not in S^*;
(5) $\mathcal{S}' = \mathrm{sc}(x_1^* x_2^* \cdots x_k^*)$;

Clearly not every combination results in a sensible question to study. In order to study $\mathcal{L}$, the length of the longest word omitted by S^*, we clearly need to impose condition (ii), that S^* be co-finite.

We now study under what conditions it makes sense to study $\mathcal{K} = \max_{x \in \Sigma^* - x_1^* x_2^* \cdots x_k^*} |x|$, the length of the longest word not in $x_1^* x_2^* \cdots x_k^*$.

Theorem 3.1. *Let $x_1, x_2, \ldots, x_k \in \Sigma^+$. Then $Q = x_1^* x_2^* \cdots x_k^*$ is co-finite if and only if $|\Sigma| = 1$ and $\gcd(|x_1|, \ldots, |x_k|) = 1$.*

Proof. If $|\Sigma| = 1$ and $\gcd(|x_1|, \ldots, |x_k|) = 1$, then every sufficiently long unary word can be obtained by concatenations of the x_i, so Q is co-finite.

For the other direction, suppose Q is co-finite. If $|\Sigma| = 1$, let $\gcd(|x_1|, \ldots, |x_k|) = d$. If $d > 1$, Q contains only words of length divisible by d, and so is not co-finite. So $d = 1$.

Hence assume $|\Sigma| \geq 2$, and let a, b be distinct letters in Σ. Let $l = \max_{1 \leq i \leq k} |x_i|$, the length of the longest word. Let $Q' = ((a^{2l} b^{2l})^k)^+$. Then we claim that $Q' \cap Q = \emptyset$. For if none of the x_i consists of powers of a single letter, then the longest block of consecutive identical letters in any word in Q is $< 2l$, so no word in Q' can be in Q. Otherwise, say some of the x_i consist of powers of a single letter. Take any word w in Q, and count the number $n(w)$ of maximal blocks of $2l$ or more consecutive identical letters in w. Clearly $n(w) \leq k$. But $n(w') \geq 2k$ for any word w' in Q'. Thus Q is not co-finite, as it omits all the words in Q'. ∎

4. State complexity results

In this section we study the measures $\mathcal{S} = \mathrm{sc}(S^*)$, and $\mathcal{S}' = \mathrm{sc}(x_1^* x_2^* \cdots x_k^*)$. First we review previous results.

Yu, Zhuang, and Salomaa [27] showed that if L is accepted by a DFA with n states, then L^* can be accepted by a DFA with at most $2^{n-1} + 2^{n-2}$ states. Furthermore, they showed this bound is realized, in the sense that for all $n \geq 2$, there exists a DFA M with n states such that the minimal DFA accepting $L(M)^*$ needs $2^{n-1} + 2^{n-2}$ states. This latter result was given previously by Maslov [21].

Câmpeanu et al. [3, 5] showed that if a DFA with n states accepts a *finite* language L, then L^* can be accepted by a DFA with at most $2^{n-3} + 2^{n-4}$ states for $n \geq 4$. Furthermore, this bound is actually achieved for $n > 4$ for an alphabet of size 3 or more. Unlike the examples we are concerned with in this section, however, the finite languages they construct contain exponentially many words in n.

Holzer and Kutrib [16] examined the nondeterminstic state complexity of Kleene star. They showed that if an NFA M with n states accepts L, then L^* can be accepted by an NFA with $n + 1$ states, and this bound is tight. If L is finite, then $n - 1$ states suffices, and this bound is tight.

Câmpeanu and Ho [4] gave tight bounds for the number of states required to accept a finite language whose words are all bounded by length n.

Proposition 4.1.

(a) $\mathrm{nsc}(\{x_1, x_2, \ldots, x_k\}^*) \leq m - k + 1$.
(b) $\mathrm{sc}(\{x_1, x_2, \ldots, x_k\}^*) \leq 2^{m-k+1}$.
(c) *If no x_i is a prefix of any other x_j, then* $\mathrm{sc}(\{x_1, x_2, \ldots, x_k\}^*) \leq m - k + 2$.

We now recall an example providing a lower bound for the state complexity of $\{x_1, x_2, \ldots, x_k\}^*$ [13]. Let t be an integer ≥ 2, and define words as follows: $y := \mathtt{0}1^{t-1}\mathtt{0}$ and $x_i := 1^{t-i-1}\mathtt{0}1^{i+1}$ for $0 \leq i \leq t-2$. Let $S_t := \{\mathtt{0}, x_0, x_1, \ldots, x_{t-2}, y\}$.

Theorem 4.2. *S_t^* has state complexity $3t2^{t-2} + 2^{t-1}$.*

Corollary 4.3. *There exists a family of sets S_t, each consisting of $t+1$ words of length $\leq t+1$, such that $\mathrm{sc}(S_t^*) = 2^{\Omega(t)}$. If m is the total number of symbols in these words, then $\mathrm{sc}(S_t^*) = 2^{\Omega(\sqrt{m})}$.*

Using similar ideas, we can also create an example achieving subexponential state complexity for $x_1^* x_2^* \cdots x_k^*$.

Theorem 4.4. *Let y and x_i be as defined above. Let $L = (\mathtt{0}^* x_1^* x_2^* \cdots x_{n-1}^* y^*)^e$ where $e = (t+1)(t-2)/2 + 2t$. Then $\mathrm{sc}(L) \geq 2^{t-2}$.*

Proof. Define $A = \{x_0, x_1, \ldots, x_{t-2}, y, \mathtt{0}\}$ and $T = \{x_1, x_2, \ldots, x_{t-2}\}$. For any subset S of T, say $\{s_1, s_2, \ldots, s_j\}$ with $s_1 < s_2 < \cdots s_j$ define

$$x(S) = y x_{t-2} y x_{t-3} x_{t-2} y \cdots y x_1 x_2 \cdots x_{t-2} y x_{s_1} x_{s_2} \cdots x_{s_j} y.$$

Note that $x(S)$ contains t copies of y and at most $(t-2)(t-1)/2 + t - 2 = (t+1)(t-2)/2$ x's. Thus $|x(S)| \leq (t+1)(t + (t+1)(t-2)/2)$ and $|x(S)|_0 \leq 2t + (t+1)(t-2)/2$.

To get the bound $\mathrm{sc}(L) \geq 2^{t-2}$, we exhibit 2^{t-2} pairwise distinct words under the Myhill-Nerode equivalence relation. Let R and S be two distinct subsets of T, and without loss of generality, let $m \in R$, $m \notin S$. By the proof of [13, Theorem 13] we have $x(R)1^{t-m} \in A^*$ but $x(S)1^{t-m} \notin A^*$. Since $L \subseteq A^*$, $x(S)1^{t-m} \notin L$. It remains to see $x(R)1^{t-m} \in L$.

Since $x(R)1^{t-m} \in A^*$, there exists a factorization of $x(R)1^{t-m}$ in terms of elements of A. However, $|x(R)1^{t-m}| \leq |x(R)| + t \leq (t+1)(t + (t+1)(t-2)/2 + t)$ so any factorization of $x(R)1^{t-m}$ into elements of A contains at most $(t+1)(t-2)/2 + 2t$ copies of words other than 0. Similarly $|x(R)1^{t-m}|_0 \leq |x(R)| \leq (t+1)(t-2)/2 + 2t$, so any factorization of $x(R)1^{t-m}$ into elements of A contains at most $(t+1)(t-2)/2 + 2t$ copies of the word 0. Thus a factorization of $x(R)1^{t-m}$ into elements of A is actually contained in L. ∎

Corollary 4.5. *There exists an infinite family of tuples $(x_1, x_2, \ldots, x_k)$ where m, the total number of symbols, is $O(t^4)$, and $\mathrm{sc}(x_1^* \cdots x_k^*) = 2^{\Omega(t)} = 2^{\Omega(m^{1/4})}$.*

We now turn to an upper bound on the state complexity of S^* in the case where the number of words in S is not specified, but we do have a bound on the length of the longest word.

Theorem 4.6. *Let $S = \{x_1, x_2, \ldots, x_k\}$ be a finite set with $\max_{1 \leq i \leq k} |x_i| = n$, that is, the longest word is of length n. Then $\mathrm{sc}(S^*) \leq \frac{2}{2|\Sigma|-1}(2^n|\Sigma|^n - 1)$.*

Proof. The idea is to create a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that records the last $n-1$ symbols seen, together with the set of the possible positions inside those $n-1$ symbols where the factorization of the input into elements of S could end.

The number of states in this DFA is $\sum_{0 \leq i < n} |\Sigma|^i 2^{i+1} = \frac{2}{2|\Sigma|-1}(2^n|\Sigma|^n - 1)$. ∎

5. State complexity for two words

In this section we develop formulas bounding the state complexity of $\{w, x\}^*$ and w^*x^*. Here, as usual, $g(x_1, x_2)$ denotes the Frobenius function introduced in Section 1. We need the following lemma, which is of independent interest and which generalizes a classical theorem of Fine and Wilf [14].

Lemma 5.1. *Let w and x be nonempty words. Let $y \in w\{w, x\}^\omega$ and $z \in x\{w, x\}^\omega$. Then the following conditions are equivalent:*

(a) *y and z agree on a prefix of length $|w| + |x| - \gcd(|w|, |x|)$;*
(b) *$wx = xw$;*
(c) *$y = z$.*

Furthermore, the bound in (a) is optimal, in the sense that for all pairs of lengths (m, n) there exist w, x with $(m, n) = (|w|, |x|)$ such that w^ω and x^ω agree on a prefix of length $|w| + |x| - \gcd(|w|, |x|) - 1$.

Proof. (a) $\implies$ (b): We prove the contrapositive. Suppose $wx \neq xw$. Then we prove that y and z differ at a position $\leq |w| + |x| - \gcd(|w|, |x|)$. The proof is by induction on $|w| + |x|$. The base case is $|w| = |x| = 1$ and is left to the reader.

Now assume the result is true for $|w| + |x| < k$. We prove it for $|w| + |x| = k$. If $|w| = |x|$ then y and z must disagree at the $|w|$'th position or earlier, for otherwise $w = x$ and $wx = xw$; since $|w| \leq d = |w|$, the result follows. So, without loss of generality, assume $|w| < |x|$. If w is not a prefix of x, then y and z disagree at the $|w|$'th position or earlier, and again $|w| \leq d$.

So w is a proper prefix of x. Write $x = wt$ for some nonempty word t. Now any common divisor of $|w|$ and $|x|$ must also divide $|x| - |w| = |t|$, and similarly any common divisor of both $|w|$ and $|t|$ must also divide $|w| + |t| = |x|$. So $\gcd(|w|, |x|) = \gcd(|w|, |t|)$.

Now $wt \neq tw$, for otherwise we have $wx = wwt = wtw = xw$, a contradiction. Then y begins with ww and z begins with wt. By induction (since $|w| + |t| < k$) $w^{-1}y$ and $w^{-1}z$ disagree at position $|w| + |t| - \gcd(|w|, |t|)$ or earlier. Hence y and z disagree at position $2|w| + |t| - \gcd(|w|, |t|) = d$ or earlier.

(b) $\implies$ (c): If $wx = xw$, then by the theorem of Lyndon-Schützenberger, both w and x are powers of a common word u. Hence $y = u^\omega = z$.

(c) $\implies$ (a): Trivial.

For the optimality statement, the words constructed in the paper [6] suffice. ∎

Theorem 5.2. *Let $w, x \in \Sigma^+$. Then*

$$\mathrm{sc}(\{w, x\}^*) \leq \begin{cases} |w| + |x|, & \text{if } wx \neq xw; \\ d\,(g(|w|/d, |x|/d) + 1) + 2, & \text{if } wx = xw \text{ and } d = \gcd(|w|, |x|)\ . \end{cases}$$

Proof. If $wx = xw$, then by a classical theorem of Lyndon and Schützenberger [20], we know there exists a word z and integers $i, j \geq 1$ such that $w = z^i$, $x = z^j$. Thus $\{w, x\}^* = \{z^i, z^j\}^*$. Let $e = \gcd(i, j)$. Then $L = \{z^i, z^j\}^*$ consists of all words of the form z^{ke} for $k > g(i/e, j/e)$, together with some words of the form z^{ke} for $0 \leq k < g(i/e, j/e)$. Thus, as in the proof of Corollary 2.3, we can accept L with a "tail" of $e|z|g(i/e, j/e) + 1$ states and a "loop" of $e|z|$ states. Adding an additional state as a "dead state" to absorb unused transitions gives a total of $(e|z|(g(i/e, j/e)+1)+2$ states. Since $d = e|z|$, the bound follows.

Otherwise, $xw \neq wx$. Without loss of generality, let us assume that $|w| \leq |x|$. Suppose w is not a prefix of x. Let p be the longest common prefix of w and x. Then we can write $w = paw'$ and $x = pbx'$ for $a \neq b$. Then we can accept $\{w, x\}^*$ with a transition diagram that has one chain of nodes labeled p leading from q_0 to a state q, and two additional chains leading from q back to q_0, one labled aw' and one labeled bx'. Since $a \neq b$, this is a DFA. One additional "dead state" might be required to absorb transitions on letters not mentioned. The total number of states is $|p| + 1 + |w'| + |x'| + 1 \leq |w| + |x|$.

Finally, suppose $|w| \leq |x|$ and w is a prefix of x. We claim it suffices to bound the longest common prefix between any word of $w\{w, x\}^*$ and $x\{w, x\}^*$. For if the longest common prefix is of length b, we can distinguish between them after reading $b + 1$ symbols. The $b+1$'th symbol must be one of two possibilities, and we can use back arrows in the transition diagram to the appropriate state. We may need one additional state as a "dead state", so the total number of states needed is $b + 2$. But from Lemma 5, we know $b \leq |w| + |x| - 2$. ∎

Theorem 5.3. *Let $w, x \in \Sigma^+$. Then*

$$\mathrm{sc}(w^*x^*) \leq \begin{cases} |w| + 2|x|, & \text{if } wx \neq wx; \\ d(g(|w|/d, |x|/d) + 1) + 2, & \text{if } wx = xw \text{ and } d = \gcd(|w|, |x|) \ . \end{cases}$$

Proof. Similar to the proof of the previous theorem. Omitted. ∎

6. Longest word omitted

In this section we assume that $S = \{x_1, x_2, \ldots, x_k\}$ for finite words $x_1, x_2, \ldots, x_k$, and S^* is co-finite. We first obtain an upper bound on the length of the longest word not in S^*.

Theorem 6.1. *Suppose $|x_i| \leq n$ for all i. Then if S^* is co-finite, the length of the longest word not in S^* is $< \frac{2}{2|\Sigma|-1}(2^n|\Sigma|^n - 1)$.*

In the rest of this section we show that the length of the longest word not in S^* can be exponentially long in n. We need several preliminary results first.

We say that x is a *proper prefix* of a word y if $y = xz$ for a nonempty word z. Similarly, we say x is a *proper suffix* of y if $y = zx$ for a nonempty word z.

Proposition 6.2. *Let S be a finite set of nonempty words such that S^* is co-finite, and $S^* \neq \Sigma^*$. Then for all $x \in S$, there exists $x' \in S$ such that x is a proper prefix of x, or vice versa. Similarly, for all $x \in S$, there exists $x' \in S$ such that x is a proper suffix of x', or vice versa.*

Proof. Let $x \in S$. Since $S^* \neq \Sigma^*$, there exists $v \in \overline{S^*}$. Since S^* is co-finite, $S^* \cap x^*v$ is nonempty. Let $i \geq 0$ be the smallest integer such that $x^iv \in S^*$; then $i \geq 1$, for otherwise $v \in S^*$. Since $x^iv \in S^*$, there exist $y_1, y_2, \ldots, y_j \in S$ such that $x^iv = y_1y_2 \ldots y_j$. Now $y_1 \neq x$, for otherwise by cancelling an x from both sides, we would have $x^{i-1}v \in S^*$, contradicting the minimality of i. If $|x| < |y_1|$, then x is a proper prefix of y_1, while if $|x| > |y_1|$, then y_1 is a proper prefix of x.

A similar argument applies for the result about suffixes. ∎

Next, we give two lemmas that characterize those sets S such that S^* is co-finite, when S is a set containing words of no more than two distinct lengths.

Lemma 6.3. *Suppose $S \subseteq \Sigma^m \cup \Sigma^n$, $0 < m < n$, and S^* is co-finite. Then $\Sigma^m \subseteq S$.*

Proof. If $S^* = \Sigma^*$, then S must contain every word x of length m, for otherwise S^* would omit x. So assume $S^* \neq \Sigma^*$.

Let $x \in \Sigma^m$. Then $S^* \cap x\Sigma^*$ is nonempty, since S^* is co-finite. Choose v such that $xv \in S^*$; then there is a factorization $xv = y_1 y_2 \cdots y_j$ where each $y_i \in S$. If $y_1 \in \Sigma^m$, then $x = y_1$ and so $x \in S$. Otherwise $y_1 \in \Sigma^n$. By Proposition 6.2, there exists $z \in S$ such that y_1 is a proper prefix of z or vice versa. But since S contains words of only lengths m and n, and $y_1 \in \Sigma^n$, we must have $z \in \Sigma^m$, and z is a prefix of y_1. Then $x = z$, and so $x \in S$. ∎

Lemma 6.4. *Suppose $S \subseteq \Sigma^m \cup \Sigma^n$, with $0 < m < n < 2m$ and S^* is co-finite. Then $\Sigma^l \subseteq S^*$, where $l = m|\Sigma|^{n-m} + n - m$.*

Proof. Let x be a word of length l that is not in S^*. Then we can write x uniquely as

$$x = y_0 z_0 y_1 z_1 \cdots y_{|\Sigma|^{n-m}-1} z_{|\Sigma|^{n-m}-1} y_{|\Sigma|^{n-m}}, \tag{6.1}$$

where $y_i \in \Sigma^{n-m}$ for $0 \leq i \leq |\Sigma|^{n-m}$, and $z_i \in \Sigma^{2m-n}$ for $0 \leq i < |\Sigma|^{n-m}$.

Now suppose that $y_i z_i y_{i+1} \in S$ for some i with $0 \leq i < |\Sigma|^{n-m}$. Then we can write

$$x = \left(\prod_{0 \leq j < i} y_j z_j \right) y_i z_i y_{i+1} \left(\prod_{i+1 \leq k \leq |\Sigma|^{n-m}} z_k y_k \right).$$

Note that $|y_j z_j| = |z_k y_k| = m$. From Lemma 6.3, each term in this factorization is in S. Hence $x \in S^*$, a contradiction. It follows that

$$y_i z_i y_{i+1} \notin S \text{ for all } i \text{ with } 0 \leq i < |\Sigma|^{n-m}. \tag{6.2}$$

Now the factorization of x in Eq. (6.1) uses $|\Sigma|^{n-m} + 1$ y's, and there are only $|\Sigma|^{n-m}$ distinct words of length $n - m$. So, by the pigeonhole principle, we have $y_p = y_q$ for some $0 \leq p < q \leq |\Sigma|^{n-m}$. Now define

$$\begin{aligned} u &= y_0 z_0 \cdots y_{p-1} z_{p-1} \\ v &= y_p z_p \cdots y_{q-1} z_{q-1} \\ w &= y_q z_q \cdots y_{|\Sigma|^{n-m}}, \end{aligned}$$

so $x = uvw$. Since S^* is co-finite, there exists a smallest exponent $k \geq 0$ such that $uv^k w \in S^*$.

Now let $uv^k w = x_1 x_2 \cdots x_j$ be a factorization into elements of S. Then x_1 is a word of length m or n. If $|x_1| = n$, then comparing lengths gives $x_1 = y_0 z_0 y_1$. But by (6.2) we know $y_0 z_0 y_1 \notin S$. So $|x_1| = m$, and comparing lengths gives $x_1 = y_0 z_0$. By similar reasoning we see that $x_2 = y_1 z_1$, and so on. Hence $x_j = y_{|\Sigma|^{n-m}-1} z_{|\Sigma|^{n-m}-1} y_{|\Sigma|^{n-m}} \in S$. But this contradicts (6.2).

Thus, our assumption that $x \notin S^*$ must be false, and so $x \in S^*$. Since x was arbitrary, this proves the result. ∎

Now we can prove an upper bound on the length of omitted words, in the case where S contains words of at most two distinct lengths.

Theorem 6.5. *Suppose $S \subseteq \Sigma^m \cup \Sigma^n$, where $0 < m < n < 2m$, and S^* is co-finite. Then the length of the longest word not in S^* is $\leq g(m,l) = ml - m - l$, where $l = m|\Sigma|^{n-m} + n - m$.*

Proof. Any word in S^* must be a concatenation of words of length m and n. If $\gcd(m,n) = d > 1$, then S^* omits all words whose length is not congruent to 0 (mod d), so S^* is not co-finite, contrary to the hypothesis. Thus $\gcd(m,n) = 1$.

By Lemmas 6.3 and 6.4, we have $\Sigma^m \cup \Sigma^l \subseteq S^*$, where $l = m|\Sigma|^{n-m} + n - m$. Hence S^* contains all words of length m and l; since $\gcd(m,l) = 1$, S^* contains all words of length $> g(m,l)$. ∎

Remark. We can actually improve the result of the previous theorem to arbitrary m and n, thus giving an upper bound in the case where S consists of words of exactly two distinct lengths. Details will appear in a later version of the paper.

Corollary 6.6. *Suppose $S \subseteq \Sigma^m \cup \Sigma^n$, where $0 < m < n < 2m$ and $\gcd(m,n) = 1$. Then S^* is co-finite if and only if $\Sigma^m \subseteq S$ and $\Sigma^l \subseteq S^*$, where $l = m|\Sigma|^{n-m} + n - m$.*

Proof. If S^* is co-finite, then by Lemmas 6.3 and 6.4 we get $\Sigma^m \subseteq S$ and $\Sigma^l \subseteq S^*$. On the other hand, if $\Sigma^m \subseteq S$ and $\Sigma^l \subseteq S^*$, then since $\gcd(m,l) = 1$, every word of length $> g(m,l)$ is contained in S^*, so S^* is co-finite. ∎

We need one more technical lemma.

Lemma 6.7. *Suppose $S \subseteq \Sigma^m \cup \Sigma^n$, where $0 < m < n < 2m$, and S^* is co-finite. Let τ be a word not in S^* where $|\tau| = n + jm$ for some $j \geq 0$. Then $S^* \cap (\tau\Sigma^m)^{i-1}\tau = \emptyset$ for $1 \leq i < m$.*

Proof. As before, since S^* is co-finite we must have $\gcd(m,n) = 1$. Define $L_i = (\tau\Sigma^m)^{i-1}\tau$ for $1 \leq i < m$. We prove that $S^* \cap L_i = \emptyset$ by induction on i.

The base case is $i = 1$. Then $L_i = L_1 = \{\tau\}$. But $S^* \cap \{\tau\} = \emptyset$ by the hypothesis that $\tau \notin S^*$.

Now suppose we have proved the result for some i, $i \leq m-2$, and we want to prove it for $i+1$. First we show that $S^* \cap \Sigma^{n-m}L_i = \emptyset$. Assume that $uw \in S^*$ for some $u \in \Sigma^{n-m}$ and $w \in L_i$. Then there is a factorization

$$uw = y_1 y_2 \cdots y_t \tag{6.3}$$

where $y_h \in S$ for $1 \leq h \leq t$. Now $|uw| = n - m + (n + jm + m)(i-1) + n + jm = n(i+1) + m(ji + i - 2)$. Since $0 < i+1 < m$, m does not divide $|uw|$. Thus at least one of the y_h is of length n, for otherwise (6.3) could not be a factorization of uw into elements of S. Let r be the smallest index such that $|y_r| = n$. Then we have

$$uw = \overbrace{y_1 y_2 \cdots y_{r-1}}^{\text{all of length } m} \quad \overbrace{y_r}^{\text{of length } n} \quad y_{r+1} \cdots y_t.$$

Hence $|y_1 y_2 \cdots y_r| = m(r-1) + n = mr + n - m$. Since, by Lemma 6.3 we have $\Sigma^m \subseteq S$, we can write $y_1 \cdots y_r = u z_1 \cdots z_r$, where $z_h \in S$ for $1 \leq h \leq r$. Thus

$$\begin{aligned} uw &= y_1 \cdots y_{r-1} y_r y_{r+1} \cdots y_t \\ &= u z_1 \cdots z_r y_{r+1} \cdots y_t; \end{aligned}$$

and, cancelling the u on both sides, we get $w = z_1 \cdots z_r y_{r+1} \cdots y_t$. But each term on the right is in S, so $w \in S^*$. But this contradicts our inductive hypothesis that $S^* \cap L_i = \emptyset$.

So now we know that

$$S^* \cap \Sigma^{n-m} L_i = \emptyset; \tag{6.4}$$

we'll use this fact below.

Now assume that $S^* \cap L_{i+1} \neq \emptyset$. Since $L_{i+1} = \tau \Sigma^m L_i$, there exists $\alpha \in \Sigma^m$ and $w \in L_i$ such that $\tau \alpha w \in S^*$. Write $\tau \alpha w = g_1 g_2 \cdots g_p$, where $g_h \in S$ for $1 \leq h \leq p$. We claim that $g_h \in \Sigma^m$ for $1 \leq h \leq j+1$. For if not, let k be the smallest index such that $|g_k| = n$. Then by comparing lengths, we have

$$\tau = \overbrace{g_1 g_2 \cdots g_{k-1}}^{\text{each of length } m} \quad \overbrace{g_k}^{\text{of length } n} \quad \overbrace{g'_1 g'_2 \cdots g'_{j-k+1}}^{\text{each of length } m}$$

for some $g'_1, g'_2, \ldots, g'_{j-k+1} \in \Sigma^m$. But this shows $\tau \in S^*$, a contradiction. We also have $g_{j+1} \notin \Sigma^n$, for otherwise $\tau = g_1 \cdots g_j g_{j+1} \in S^*$, a contradiction.

Now either $g_{j+2} \in \Sigma^m$ or $g_{j+2} \in \Sigma^n$. In the former case, by comparing lengths, we see that $g_{j+3} \cdots g_p \in \Sigma^{n-m} L_i$. But this contradicts (6.4). In the latter case, by comparing lengths, we see $g_{j+3} \cdots g_p \in L_i$, contradicting our inductive hypothesis. Thus our assumption that $S^* \cap L_{i+1} \neq \emptyset$ was wrong, and the lemma is proved. ∎

Now we are ready to give a class of examples achieving the bound in Theorem 6.5. Without lose of generality, let $\Sigma = \{0, 1, \cdots\}$. We define $r(n, k, l)$ to be the word of length l representing n in base k, possibly with leading zeros. For example, $r(11, 2, 5) = \mathtt{01011}$. For integers $0 < m < n$, we define

$$T(m, n) = \{r(i, |\Sigma|, n-m)\mathtt{0}^{2m-n} r(i+1, |\Sigma|, n-m) \ : \ 0 \leq i \leq |\Sigma|^{n-m} - 2\}.$$

For example, over a binary alphabet we have $T(3, 5) = \{\mathtt{00001}, \mathtt{01010}, \mathtt{10011}\}$.

Theorem 6.8. *Let m, n be integers with $0 < m < n < 2m$ and $\gcd(m, n) = 1$, and let $S = \Sigma^m \cup \Sigma^n - T(m, n)$. Then S^* is co-finite and the longest words not in S^* are of length $g(m, l)$, where $l = m|\Sigma|^{n-m} + n - m$.*

Proof. First, let's prove that S^* is co-finite. Since $\Sigma^m \subseteq S$, by Corollary 6.6 it suffices to show that $\Sigma^l \subseteq S^*$, where $l = m|\Sigma|^{n-m} + n - m$.

Let $x \in \Sigma^l$, and write

$$x = y_0 z_0 y_1 z_1 \cdots y_{|\Sigma|^{n-m}-1} z_{|\Sigma|^{n-m}-1} y_{|\Sigma|^{n-m}}$$

where $y_i \in \Sigma^{n-m}$ for $0 \leq i \leq |\Sigma|^{n-m}$, and $z_i \in \Sigma^{2m-n}$ for $0 \leq i < |\Sigma|^{n-m}$.

If $y_i z_i y_{i+1} \in T(m, n)$ for all i, $0 \leq i < |\Sigma|^{n-m}$, then since the base-$k$ expansions are forced to match up, we have $y_i = r(i, |\Sigma|, n-m)$ for $0 \leq i < |\Sigma|^{n-m}$. But the longest such word is of length $m|\Sigma|^{n-m} + n - 2m < l$, a contradiction. Hence $y_i z_i y_{i+1} \in S$ for some i. Thus

$$x = \left(\prod_{0 \leq j < i} y_j z_j \right) y_i z_i y_{i+1} \left(\prod_{i+1 \leq k \leq |\Sigma|^{n-m}} z_k y_k \right).$$

Note that $|y_j z_j| = |z_k y_k| = m$. Since $\Sigma^m \subseteq S$, this gives a factorization of $x \in S^*$. Since x was arbitrary, we have $\Sigma^l \subseteq S^*$.

Now we will prove that $\tau \notin S^*$, where

$$\tau := r(0, |\Sigma|, n-m)\mathtt{0}^{2m-n} r(1, |\Sigma|, n-m)\mathtt{0}^{2m-n} \cdots r(|\Sigma|^{n-m} - 1, |\Sigma|, n-m).$$

Note that $|\tau| = |\Sigma|^{n-m}(n-m) + (|\Sigma|^{n-m}-1)(2m-n) = m|\Sigma|^{n-m} + n - 2m = l - m$. Suppose there exists a factorization $\tau = w_1 w_2 \cdots w_t$, where $w_i \in S$ for $1 \le i \le t$. Since $|\tau|$ is not divisible by m, at least one of these terms is of length n. Let k be the smallest index such that $w_k \in \Sigma^n$. then $\tau = w_1 \cdots w_{k-1} w_k w_{k+1} \cdots w_t$. By comparing lengths, we get $w_i = r(i-1, |\Sigma|, n-m)\mathtt{0}^{2m-n}$ for $1 \le i < k$. Thus $w_k = r(k-1, |\Sigma|, n-m)\mathtt{0}^{2m-n} r(k, |\Sigma|, n-m) \in S \cap \Sigma^n$. But $r(k-1, |\Sigma|, n-m)\mathtt{0}^{2m-n} r(k, |\Sigma|, n-m) \in T(m,n)$, a contradiction. Thus $\tau \notin S^*$.

We may now apply Lemma 6.7 to get that S^* omits words of the form $(\tau\Sigma^m)^{m-2}\tau$; these words are of length $(l-m+m)(m-2) + l - m = lm - l - m = g(m,l)$. ∎

Corollary 6.9. *For each odd integer $n \ge 5$, there exists a set of binary words of length at most n, such that S^* is co-finite and the longest word not in S^* is of length $\Omega(n^2 2^{n/2})$.*

Proof. Choose $m = (n+1)/2$ and apply Theorem 6.8. ∎

Example 6.10. Let $m = 3$, $n = 5$, $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. Then $S = \Sigma^3 + \Sigma^5 - \{\mathtt{00001}, \mathtt{01010}, \mathtt{10011}\}$. Then a longest word not in S^* is $\mathtt{00001010011\,000\,00001010011}$, of length 25.

7. Number of omitted words

Recall that $f(x_1, x_2, \ldots, x_k)$ is the classical function which, for positive integers $x_1, \ldots, x_k$ with $\gcd(x_1, \ldots, x_k) = 1$, counts the number of integers not representable as a non-negative integer linear combination of the x_i. In this section we consider a generalization of this function to the setting of a free monoid, replacing the integers x_i with finite words in Σ^*, and replacing the condition $\gcd(x_1, \ldots, x_k) = 1$ with the requirement that $\{x_1, \ldots, x_k\}^*$ be co-finite.

We have already studied this in the case of a unary alphabet in Section 2, so let us assume that Σ has at least two letters.

Theorem 7.1. *Let $x_1, x_2, \ldots, x_k \in \Sigma^*$ be such that $|x_i| \le n$ for $1 \le i \le n$. Let $S = \{x_1, x_2, \ldots, x_k\}$ and suppose S^* is co-finite. Then*

$$\mathcal{M} = |\Sigma^* - S^*| \le \frac{|\Sigma|^q - 1}{|\Sigma| - 1},$$

where $q = \frac{2}{2|\Sigma|-1}(2^n|\Sigma|^n - 1)$.

Proof. From Theorem 6.1, we know that if S^* is co-finite, the length of the longest omitted word is $< q$, where $q = \frac{2}{2|\Sigma|-1}(2^n|\Sigma|^n - 1)$. The total number of words $< q$ is $1 + |\Sigma| + \cdots + |\Sigma|^{q-1} = \frac{|\Sigma|^q - 1}{|\Sigma|-1}$. ∎

We now give an example achieving a doubly-exponential lower bound on $\mathcal{M}$.

Theorem 7.2. *Let m, n be integers with $0 < m < n < 2m$ and $\gcd(m,n) = 1$, and let $S = \Sigma^m \cup \Sigma^n - U(m,n)$, where U is defined by*

$$U(m,n) = \{r(i, |\Sigma|, n-m)\mathtt{0}^{2m-n} r(j, |\Sigma|, n-m) \ : \ 0 \le i < j \le |\Sigma|^{n-m} - 1\}.$$

Then S^* is co-finite and S^* omits at least $2^{|\Sigma|^{n-m}} - |\Sigma|^{n-m} - 1$ words.

Proof. Similar to that of Theorem 6.8. ∎

References

[1] M. Beck, R. Diaz, and S. Robins. The Frobenius problem, rational polytopes, and Fourier-Dedekind sums. *J. Number Theory* **96** (2002), 1–21.

[2] A. Brauer. On a problem of partitions. *Amer. J. Math.* **64** (1942), 299–312.

[3] C. Câmpeanu, K. Culik II, K. Salomaa, and S. Yu. State complexity of basic operations on finite languages. In *WIA'99*, *Lect. Notes in Comp. Science* 2214, pp. 60-70, 2001.

[4] C. Câmpeanu and W. H. Ho. The maximum state complexity for finite languages. *J. Automata, Languages, and Combinatorics* **9** (2004), 189–202.

[5] C. Câmpeanu, K. Salomaa, and S. Yu. State complexity of regular languages: finite versus infinite. In C. S. Calude and G. Păun, eds., *Finite Versus Infinite: Contributions to an Eternal Dilemma*, pp. 53–73. Springer, 2000.

[6] S. Cautis, F. Mignosi, J. Shallit, M.-w. Wang, and S. Yazdani. Periodicity, morphisms, and matrices. *Theoret. Comput. Sci.* **295** (2003), 107–121.

[7] M. Chrobak. Finite automata and unary languages. *Theoret. Comput. Sci.* **47** (1986), 149–158. Errata, **302** (2003), 497–498.

[8] P. Chrzastowski-Wachtel and M. Raczunas. Liveness of weighted circuits and the Diophantine problem of Frobenius. In Z. Ésik, ed., *FCT '93*, *Lect. Notes in Comp. Science* 710, pp. 171–180, 1993.

[9] J. Clément, J.-P. Duval, G. Guaina, D. Perrin, and G. Rindone. Parsing with a finite dictionary. *Theoret. Comput. Sci.* **340** (2005), 432–442.

[10] J. L. Davison. On the linear diophantine problem of Frobenius. *J. Number Theory* **48** (1994), 353–363.

[11] A. L. Dulmage and N. S. Mendelsohn. Gaps in the exponent set of primitive matrices. *Illinois J. Math.* **8** (1964), 642–656.

[12] D. Einstein, D. Lichtblau, A. Strzebonski, and S. Wagon. Frobenius numbers by lattice point enumeration. *Integers* **7** (2007), A15 (electronic).

[13] K. Ellul, B. Krawetz, J. Shallit, and M.-w. Wang. Regular expressions: new results and open problems. *J. Autom. Lang. Combin.* **10** (2005), 407–437.

[14] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.* **16** (1965), 109–114.

[15] H. Greenberg. Solution to a linear Diophantine equation for nonnegative integers. *J. Algorithms* **9** (1988), 343–353.

[16] M. Holzer and M. Kutrib. Nondeterministic descriptional complexity of regular languages. *Internat. J. Found. Comp. Sci.* **14** (2003), 1087–1102.

[17] J. Incerpi and R. Sedgewick. Improved upper bounds on shellsort. *J. Comput. System Sci.* **31** (1985), 210–224.

[18] R. Kannan. Lattice translates of a polytope and the Frobenius problem. *Combinatorica* **12** (1992), 161–177.

[19] E. Kunz. The value-semigroup of a one-dimensional Gorenstein ring. *Proc. Amer. Math. Soc.* **25** (1970), 748–751.

[20] R. C. Lyndon and M. P. Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Math. J.* **9** (1962), 289–298.

[21] A. N. Maslov. Estimates of the number of states of finite automata. *Dokl. Akad. Nauk. SSSR* **194** (1970), 1266–1268. In Russian. English translation in *Soviet Math. Dokl.* **11** (1970), 1373–1375.

[22] J. L. Ramírez-Alfonsín. Complexity of the Frobenius problem. *Combinatorica* **16** (1996), 143–147.

[23] J. L. Ramírez-Alfonsín. *The Diophantine Frobenius Problem*. Oxford University Press, 2005.

[24] J. J. Sylvester. Problem 7382. *Math. Quest. Sol. Educ. Times* **41** (1884), ix, 21.

[25] E. Teruel, P. Chrzastowski-Wachtel, J. M. Colom, and M. Silva. On weighted T-systems. In K. Jensen, editor, *Applic. and Theory of Petri Nets 1992*, *Lect. Notes in Comp. Science* 616, pp. 348–367, 1992.

[26] H. S. Wilf. A circle-of-lights algorithm for the "money-changing problem". *Amer. Math. Monthly* **85** (1978), 562–565.

[27] S. Yu, Q. Zhuang, and K. Salomaa. The state complexities of some basic operations on regular languages. *Theoret. Comput. Sci.* **125** (1994), 315–328.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 433-444
www.stacs-conf.org

SPACE HIERARCHY RESULTS FOR RANDOMIZED MODELS

JEFF KINNE [1] AND DIETER VAN MELKEBEEK [1]

[1] Department of Computer Sciences, University of Wisconsin-Madison, USA
E-mail address: {jkinne,dieter}@cs.wisc.edu

Abstract. We prove space hierarchy and separation results for randomized and other semantic models of computation with advice. Previous works on hierarchy and separation theorems for such models focused on time as the resource. We obtain tighter results with space as the resource. Our main theorems are the following. Let $s(n)$ be any space-constructible function that is $\Omega(\log n)$ and such that $s(an) = O(s(n))$ for all constants a, and let $s'(n)$ be any function that is $\omega(s(n))$.

There exists a language computable by *two-sided* error randomized machines using $s'(n)$ space and one bit of advice that is not computable by *two-sided* error randomized machines using $s(n)$ space and $\min(s(n), n)$ bits of advice.

There exists a language computable by *zero-sided* error randomized machines in space $s'(n)$ with one bit of advice that is not computable by *one-sided* error randomized machines using $s(n)$ space and $\min(s(n), n)$ bits of advice.

The condition that $s(an) = O(s(n))$ is a technical condition satisfied by typical space bounds that are at most linear. We also obtain weaker results that apply to generic semantic models of computation.

1. Introduction

A hierarchy theorem states that the power of a machine increases with the amount of resources it can use. Time hierarchy theorems on deterministic Turing machines follow by direct diagonalization: a machine N diagonalizes against every machine M_i running in time t by choosing an input x_i, simulating $M_i(x_i)$ for t steps, and then doing the opposite. Deriving a time hierarchy theorem for nondeterministic machines is more complicated because a nondeterministic machine cannot easily complement another nondeterministic machine (unless NP=coNP). A variety of techniques can be used to overcome this difficulty, including translation arguments and delayed diagonalization [4, 13, 16].

In fact, these techniques allow us to prove time hierarchy theorems for just about any *syntactic* model of computation. We call a model syntactic if there exists a computable

1998 ACM Subject Classification: F.1.3.

Key words and phrases: Computations with Advice, Space Hierarchy, Randomized Machine, Promise Classes, Semantic Models.

First author supported and second author partially supported by NSF Career award CCR-0133693.

enumeration of all machines in the model. For example, we can enumerate all nondeterministic Turing machines by representing their transition functions as strings and then iterating over all such strings to discover each nondeterministic Turing machine.

Many models of computation of interest are not syntactic, but *semantic*. A semantic model is defined by imposing a promise on a syntactic model. A machine belongs to the model if it is output by the enumeration of the underlying syntactic model and its execution satisfies the promise on every input. Bounded-error randomized Turing machines are an example of a non-syntactic semantic model. There does not exist a computable enumeration consisting of exactly all randomized Turing machines that satisfy the promise of bounded error on every input, but we can enumerate all randomized Turing machines and attempt to select among them those that have bounded error. In general promises make diagonalization problematic because the diagonalizing machine must satisfy the promise everywhere but has insufficient resources to determine whether a given machine from the enumeration against which it tries to diagonalize satisfies the promise on a given input.

Because of these difficulties there has yet to be a single non-trivial proof of a time hierarchy theorem for any non-syntactic model. A recent line of research [1, 5, 7, 6, 11] has provided progress toward proving time hierarchy results for non-syntactic models, including two-sided error randomized machines. Each of these results applies to semantic models that take advice, where the diagonalizing machine is only guaranteed to satisfy the promise when it is given the correct advice. Many of the results require only one bit of advice, which the diagonalizing machine uses to avoid simulating a machine on an input for which that machine breaks the promise.

As opposed to the setting of time, fairly good space hierarchy theorems are known for certain non-syntactic models. In fact, the following simple translation argument suffices to show that for any constant $c > 1$ there exists a language computable by two-sided error randomized machines using $(s(n))^c$ space that is not computable by such machines using $s(n)$ space [10], for any space-constructible $s(n)$ that is $\Omega(\log n)$. Suppose by way of contradiction that every language computable by two-sided error machines in space $(s(n))^c$ is also computable by such machines in space $s(n)$. A padding argument then shows that in that model any language computable in $(s(n))^{c^2}$ space is computable in space $(s(n))^c$ and thus in space $s(n)$. We can iterate this padding argument any constant number of times and show that for any constant d, any language computable by two-sided error machines in space $(s(n))^d$ is also computable by such machines in $s(n)$ space. For $d > 1.5$ we reach a contradiction with the deterministic space hierarchy theorem because randomized two-sided error computations that run in space $s(n)$ can be simulated deterministically in space $(s(n))^{1.5}$ [12]. The same argument applies to non-syntactic models where $s(n)$ space computations can be simulated deterministically in space $(s(n))^d$ for some constant d, including one- and zero-sided error randomized machines, unambiguous machines, etc.

Since we can always reduce the space usage by a constant factor by increasing the work-tape alphabet size, the tightest space hierarchy result one might hope for is to separate space $s'(n)$ from space $s(n)$ for any space-constructible function $s'(n) = \omega(s(n))$. For models like nondeterministic machines, which are known to be closed under complementation in the space-bounded setting [8, 14], such tight space hierarchies follow by straightforward diagonalization. For generic syntactic models, tight space hierarchies follow using the same techniques as in the time-bounded setting. Those techniques all require the existence of an

efficient universal machine, which presupposes the model to be syntactic. For that reason they fail for non-syntactic models of computation such as bounded-error machines.

In this paper we obtain space hierarchy results that are tight with respect to space by adapting to the space-bounded setting techniques that have been developed for proving hierarchy results for semantic models in the time-bounded setting. Our results improve upon the space hierarchy results that can be obtained by the simple translation argument.

1.1. Our Results

Space hierarchy results have a number of parameters: (1) the gap needed between the two space bounds, (2) the amount of advice that is needed for the diagonalizing machine N, (3) the amount of advice that can be given to the smaller space machines M_i, and (4) the range of space bounds for which the results hold. We consider (1) and (2) to be of the highest importance. We focus on space hierarchy theorems with an optimal separation in space – where any super-constant gap in space suffices. The ultimate goal for (2) is to remove the advice altogether and obtain uniform hierarchy results. As in the time-bounded setting, we do not achieve this goal but get the next best result – a single bit of advice for N suffices in each of our results. Given that we strive for space hierarchies that are tight with respect to space and require only one bit of advice for the diagonalizing machine, we aim to optimize the final two parameters.

1.1.1. *Randomized Models.* Our strongest results apply to randomized models. For two-sided error machines, we can handle a large amount of advice and any typical space bound between logarithmic and linear. We point out that the latter is an improvement over results in the time-bounded setting, in the sense that there tightness degrades for all super-polynomial time bounds whereas here the results remain tight for a range of space bounds.

Theorem 1.1. *Let $s(n)$ be any space-constructible function that is $\Omega(\log n)$ and such that $s(an) = O(s(n))$ for all constants a, and let $s'(n)$ be any function that is $\omega(s(n))$. There exists a language computable by two-sided error randomized machines using $s'(n)$ space and one bit of advice that is not computable by two-sided error randomized machines using $s(n)$ space and $\min(s(n), n)$ bits of advice.*

For $s(n) = \log(n)$, Theorem 1.1 gives a bounded-error machine using only slightly larger than $\log n$ space that uses one bit of advice and differs from all bounded-error machines using $O(\log n)$ space and $O(\log n)$ bits of advice. The condition that $s(an) = O(s(n))$ for all constants a is a technical condition needed to ensure the construction yields a tight separation in space. The condition is true of all natural space bounds that are at most linear. More generally, our construction works for arbitrary space bounds $s(n)$ and space-constructible $s'(n)$ such that $s'(n) = \omega(s(n + as(n)))$ for all constants a.

Our second result gives a separation result with similar parameters as those of Theorem 1.1 but for the cases of one- and zero-sided error randomized machines. We point out that the separation result for zero-sided error machines is new to the space-bounded setting as the techniques used to prove stronger separations in the time-bounded setting do not work for zero-sided error machines. In fact, we show a single result that captures space separations for one- and zero-sided error machines – that a zero-sided error machine suffices to diagonalize against one-sided error machines.

Theorem 1.2. *Let $s(n)$ be any space-constructible function that is $\Omega(\log n)$ and such that $s(an) = O(s(n))$ for all constants a, and let $s'(n)$ be any function that is $\omega(s(n))$. There exists a language computable by zero-sided error randomized machines using $s'(n)$ space and one bit of advice that is not computable by one-sided error randomized machines using $s(n)$ space and $\min(s(n), n)$ bits of advice.*

1.1.2. *Generic Semantic Models.* The above results take advantage of specific properties of randomized machines that do not hold for arbitrary semantic models. Our final results involve a generic construction that applies to a wide class of semantic models which we term *reasonable*. We omit the precise definition due to lack of space; but besides randomized two-, one-, and zero-sided error machines, the notion also encompasses bounded-error quantum machines [15], unambiguous machines [2], Arthur-Merlin games and interactive proofs [3], etc. When applied to the logarithmic space setting, the construction gives a language computable within the model with $s'(n)$ space and one bit of advice that is not computable within the model using $O(\log n)$ space and $O(1)$ bits of advice, for any $s'(n) = \omega(\log n)$.

The performance of the generic construction is poor on the last two parameters we mentioned earlier – it allows few advice bits on the smaller space side and is only tight for $s(n) = O(\log n)$. Either of these parameters can be improved for models that can be simulated deterministically with only a polynomial blowup in space – models for which the simple translation argument works. In fact, there is a trade-off between (a) the amount of advice that can be handled and (b) the range of space bounds for which the result is tight. By maximizing the former we get the following.

Theorem 1.3. *Fix any reasonable model of computation for which space $O(\log n)$ computations can be simulated deterministically in space $O(\log^d n)$ for some rational constant d. Let $s'(n)$ be any function with $s'(n) = \omega(\log n)$. There exists a language computable using $s'(n)$ space and one bit of advice that is not computable using $O(\log n)$ space and $O(\log^{1/d} n)$ bits of advice.*

In fact, a tight separation in space can be maintained while allowing $O(\log^{1/d} n)$ advice bits for $s(n)$ any poly-logarithmic function, but the separation in space with this many advice bits is no longer tight for larger $s(n)$. By maximizing (b), we obtain a separation result that is tight for typical space bounds between logarithmic and polynomial.

Theorem 1.4. *Fix any reasonable model of computation for which space s computations can be simulated deterministically in space $O(s^d)$ for some constant d. Let $s(n)$ be a space bound that is $\Omega(\log n)$ and such that $s(n) \leq n^{O(1)}$; let $s'(n)$ be a space bound that is constructible in space $o(s'(n))$ and such that $s'(n+1) = O(s'(n))$. If $s'(n) = \omega(s(n))$ then there is a language computable in space $s'(n)$ with one bit of advice that is not computable in space $s(n)$ with $O(1)$ bits of advice.*

The first two conditions on $s'(n)$ are technical conditions true of typical space bounds in the range of interest – between logarithmic and polynomial. When applied to randomized machines, Theorem 1.4 gives a tight separation result for slightly higher space bounds than Theorems 1.1 and 1.2, but the latter can handle more advice bits.

1.2. Our Techniques

Recently, Van Melkebeek and Pervyshev [11] showed how to adapt the technique of delayed diagonalization to obtain time hierarchies for any reasonable semantic model of computation with one bit of advice. For any constant a, they exhibit a language that is computable in polynomial time with one bit of advice but not in linear time with a bits of advice. Our results for generic models of computation (Theorems 1.3 and 1.4) follow from a space-efficient implementation and a careful analysis of that approach. The proofs of these results are omitted here but included in the full paper on our web pages.

Our stronger results for randomized machines follow a different type of argument, which roughly goes as follows. When N diagonalizes against machine M_i, it tries to achieve complementary behavior on inputs of length n_i by reducing the complement of M_i at length n_i to instances of some hard language L of length somewhat larger than n_i, say m_i. N cannot compute L on those instances directly because we do not know how to compute L in small space. We instead use a delayed computation and copying scheme that forces M_i to aid N in the computation of L if M_i agrees with N on inputs larger than m_i. As a result, either M_i differs from N on some inputs larger than m_i, or else N can decide L at length m_i in small space and therefore diagonalize against M_i at length n_i.

The critical component of the copying scheme is the following task. Given a list of randomized machines with the guarantee that at least one of them satisfies the promise and correctly decides L at length m in small space, construct a single randomized machine that satisfies the promise and decides L at length m in small space. We call a procedure accomplishing this task a space-efficient *recovery procedure* for L.

The main technical contributions of this paper are the design of recovery procedures for adequate hard languages L. For Theorem 1.1 we use the computation tableau language, which is an encoding of bits of the computation tableaux of deterministic machines; we develop a recovery procedure based on the local checkability of computation tableaux. For Theorem 1.2 we use the configuration reachability language, which is an encoding of pairs of configurations that are connected in a nondeterministic machine's configuration graph; we develop a recovery procedure from the proof that NL=coNL [8, 14].

We present the basic construction for Theorems 1.1 and 1.2 with the recovery procedures as black boxes in section 3. The recovery procedure for the computation tableau language is given in section 4, and the recovery procedure for the configuration reachability language is given in section 5. Resource analysis of the construction is given in section 6.

1.2.1. *Relation to Previous Work.* Our high-level strategy is most akin to the one used in [11]. In the time-bounded setting, [11] achieves a strong separation for bounded-error randomized machines using the above construction with satisfiability as the hard language L. Hardness of L follows from the fact that randomized machines can be time-efficiently deterministically simulated using a randomized two-sided error algorithm for satisfiability. We point out that some of our results can also be obtained using a different high-level strategy than the one in [11], which can be viewed as delayed diagonalization with advice. Some of the results of [11] in the time-bounded setting can also be derived by adapting translation arguments to use advice [1, 5, 7, 6]. It is possible to derive our Theorems 1.1 and 1.2 following a space-bounded version of the latter strategy. However, the proofs still rely on the recovery procedure as a key technical ingredient and we feel that our proofs are

simpler. Moreover, for the case of generic semantic models, our approach yields results that are strictly stronger.

2. Preliminaries

We assume familiarity with standard definitions for randomized complexity classes, including two-, one-, and zero-sided error machines. For each machine model requiring randomness, we allow the machine one-way access to the randomness and only consider computations where each machine always halts in finite time.

Our separation results apply to machines that take advice. We use α and β to denote infinite sequences of advice strings. Given a machine M, M/β denotes the machine M taking advice β. Namely, on input x, M is given both x and $\beta_{|x|}$ as input. When we are interested in the execution of M/β on inputs of length n, we write M/b where $b = \beta_n$.

We consider semantic models of computation, with an associated computable enumeration $(M_i)_{i=1,2,3,\ldots}$ and an associated promise. A machine falls within the model if it is contained in the enumeration and its behavior satisfies the promise on all inputs.

For a machine M/β^* that takes advice, we only require that M satisfies the promise when given the "correct" advice sequence β^*. We note that this differs from the Karp-Lipton notion of advice of [9], where the machine must satisfy the promise no matter which advice string is given. A hierarchy for a semantic model with advice under the stronger Karp-Lipton notion would imply the existence of a hierarchy without advice.

3. Randomized Machines with Bounded Error

In this section we describe the high-level strategy used to prove Theorems 1.1 and 1.2. Most portions of the construction are the same for both, so we keep the exposition general. We aim to construct a randomized machine N and advice sequence α witnessing Theorems 1.1 and 1.2 for some space bounds $s(n)$ and $s'(n)$. N/α should always satisfy the promise, run in space $s'(n)$, and differ from M_i/β for randomized machines M_i and advice sequences β for which M_i/β *behaves appropriately*, i.e., for which M_i/β satisfies the promise and uses at most $s(n)$ space on all inputs.

As with delayed diagonalization, for each M_i we allocate an interval of input lengths $[n_i, n_i^*]$ on which to diagonalize against M_i. That is, for each machine M_i and advice sequence β such that M_i/β behaves appropriately, there is an $n \in [n_i, n_i^*]$ such that N/α and M_i/β decide differently on at least one input of length n. The construction consists of three main parts: (1) reducing the complement of the computation of M_i on inputs of length n_i to instances of a hard language L of length m_i, (2) performing a delayed computation of L at length m_i on inputs of length n_i^*, and (3) copying this behavior to smaller and smaller inputs down to input length m_i. These will ensure that if M_i/β behaves appropriately, either N/α differs from M_i/β on some input of length larger than m_i, or N/α computes L at length m_i allowing N/α to differ from M_i/b for all possible advice strings b at length n_i. We describe how to achieve (1) for two-sided error machines in section 4 and for one- and zero-sided error machines in section 5. For now, we assume a hard language L and describe (2) and (3).

Let us first try to develop the construction without assuming any advice for N or for M_i and see why N needs at least one bit of advice. On an input x of length n_i, N reduces the complement of $M_i(x)$ to an instance of L of length m_i. Because N must run in space not much more than $s(n)$ and we do not know how to compute the hard languages we use

with small space, N cannot directly compute L at length m_i. However, L can be computed at length m_i within the space N is allowed to use on much larger inputs. Let n_i^* be large enough so that L at length m_i can be deterministically computed in space $s'(n_i^*)$. We let N at length n_i^* perform a *delayed computation* of L at length m_i as follows: on inputs of the form $0^\ell y$ where $\ell = n_i^* - m_i$ and $|y| = m_i$, N uses the above deterministic computation of L on input y to ensure that $N(0^\ell y) = L(y)$.

Since N performs a delayed computation of L, M_i must as well – otherwise N already computes a language different than M_i. We would like to bring this delayed computation down to smaller padded inputs. The first attempt at this is the following: on input $0^{\ell-1}y$, N simulates $M_i(0^\ell y)$. If M_i behaves appropriately and performs the initial delayed computation, then $N(0^{\ell-1}y) = M_i(0^\ell y) = L(y)$, meaning that N satisfies the promise and performs the delayed computation of L at length m_i at an input length one smaller than before. However, M_i may not behave appropriately on inputs of the form $0^\ell y$; in particular M_i may fail to satisfy the promise, in which case N would also fail to satisfy the promise by performing the simulation. If M_i does not behave appropriately, N does not need to consider M_i and could simply abstain from the simulation. If M_i behaves appropriately on inputs of the form $0^\ell y$, it still may fail to perform the delayed computation. In that case N has already diagonalized against M_i at input length $m_i + \ell$ and can therefore also abstain from the simulation on inputs of the form $0^{\ell-1}y$.

N has insufficient resources to determine on its own if M_i behaves appropriately and performs the initial delayed computation. Instead, we give N one bit of advice at input length $m_i + \ell - 1$ indicating whether M_i behaves appropriately and performs the initial delayed computation at length $n_i^* = m_i + \ell$. If the advice bit is 0, N acts trivially at this length by always rejecting inputs. If the advice bit is 1, N performs the simulation so $N(0^{\ell-1}y)/\alpha = M_i(0^\ell y) = L(y)$.

If we give N one bit of advice, we should give M_i at least one advice bit as well. Otherwise, the hierarchy result is not fair (and is trivial). Consider how allowing M_i advice effects the construction. If there exists an advice string b such that M_i/b behaves appropriately and $M_i(0^\ell y)/b = L(y)$ for all y with $|y| = m_i$, we set N's advice bit for input length $m_i + \ell - 1$ to be 1, meaning N should copy down the delayed computation from length $m_i + \ell$ to length $m_i + \ell - 1$. Note, though, that N does not know for which advice b the machine M_i/b appropriately performs the delayed computation at length $m_i + \ell$. N has at its disposal a list of machines, M_i with each possible advice string b, with the guarantee that at least one M_i/b behaves appropriately and $M_i(0^\ell y)/b = L(y)$ for all y with $|y| = m_i$. With this list of machines as its primary resource, N wishes to ensure that $N(0^{\ell-1}y)/\alpha = L(y)$ for all y with $|y| = m_i$ while satisfying the promise and using small space.

N can accomplish this task given a space-efficient recovery procedure for L at length m_i: on input $0^{\ell-1}y$, N removes the padding and executes the recovery procedure to determine $L(y)$, for each b simulating $M_i(0^\ell y')/b$ when the recovery procedure makes a query y'. As the space complexity of the recovery procedures we give in sections 4 and 5 is within a constant factor of a single simulation of M_i, this process uses $O(s(n))$ space. We point out that for Theorem 1.1, the recovery procedure may have two-sided error, while for Theorem 1.2, the recovery procedure must have zero-sided error.

Given a recovery procedure for L, N/α correctly performs the delayed computation on inputs of length $m_i + \ell - 1$ if there is an advice string causing M_i to behave appropriately and perform the initial delayed computation at length $m_i + \ell$. We repeat the process on padded inputs of the next smaller size. Namely, N's advice bit for input length $m_i + \ell - 2$

is set to indicate if there is an advice string b such that M_i/b behaves appropriately on inputs of length $m_i + \ell - 1$ and $M_i(0^{\ell-1}y)/b = L(y)$ for all y with $|y| = m_i$. If so, then on inputs of the form $0^{\ell-2}y$, N/α uses the recovery procedure for L to determine the value of $L(y)$, for each b simulating $M_i(0^{\ell-1}y')/b$ when the recovery procedure makes a query y'. By the correctness of the recovery procedure, N/α thus correctly performs the delayed computation on padded inputs of length $m_i + \ell - 2$. If the advice bit is 0, N/α acts trivially at input length $m_i + \ell - 2$ by rejecting immediately.

We repeat the same process on smaller and smaller padded inputs. We reach the conclusion that either there is a largest input length $n \in [m_i + 1, n_i^*]$ where for no advice string b, M_i/b appropriately performs the delayed computation of L at length n; or N/α correctly computes L on inputs of length m_i. If the former is the case, N/α performs the delayed computation at length n whereas for each b either M_i/b does not behave appropriately at length n or it does but does not perform the delayed computation at length n. In either case, N/α has diagonalized against M_i/b for each possible b at length n. N's remaining advice bits for input lengths $[n_i, n-1]$ are set to 0 to indicate that nothing more needs to be done, and N/α immediately rejects inputs in this range. Otherwise N/α correctly computes L on inputs of length m_i. In that case N/α diagonalizes against M_i/b for all advice strings b at length n_i by acting as follows. On input $x_b = 0^{n_i - |b|}b$, N reduces the complement of the computation $M_i(x_b)/b$ to an instance y of L of length m_i and then simulates $N(y)/\alpha$, so $N(x_b)/\alpha = N(y)/\alpha = L(y) = \neg M_i(x_b)/b$.

We have given the major points of the construction, with the notable exception of the recovery procedures. We develop these in the next two sections. We save the resource analysis of the construction for the final section.

4. Two-sided Error Recovery Procedure – Computation Tableau Language

In this section we develop a space-efficient recovery procedure for the computation tableau language (hereafter written COMP), the hard language used in the construction of Theorem 1.1.

> COMP = $\{\langle M, x, t, j\rangle \,|\, M$ is a deterministic Turing machine, and in the t^{th} time step of executing $M(x)$, the j^{th} bit in the machine's configuration is equal to 1$\}$.

Let us see that COMP is in fact "hard" for two-sided error machines. For some input x, we would like to know whether $\Pr[M_i(x) = 1] < \frac{1}{2}$. For a particular random string, whether $M_i(x)$ accepts or rejects can be decided by looking at a single bit in M_i's configuration after a certain number of steps – by ensuring that M_i enters a unique accepting configuration when it accepts. With the randomness unfixed, we view $M_i(x)$ as defining a Markov chain on the configuration space of the machine. Provided $M_i(x)$ uses at most $s(n)$ space, a deterministic machine running in $2^{O(s)}$ time and space can estimate the state probabilities of this Markov chain to sufficient accuracy and determine whether a particular configuration bit has probability at most $1/2$ of being 1 after t time steps. This deterministic machine and a particular bit of its unique halting configuration define the instance of COMP we would like to solve when given input x.

We now present the recovery procedure for COMP. We wish to compute COMP on inputs of length m in space $O(s(m))$ with bounded error when given a list of randomized machines with the guarantee that at least one of the machines computes COMP on all

inputs of length m using $s(m)$ space with bounded error. Let $y = \langle M, x, t, j\rangle$ be an instance of COMP with $|y| = m$ that we wish to compute.

A natural way to determine COMP(y) is to consider each machine in the list one at a time and design a test that determines whether a particular machine computes COMP(y). The test should have the following properties:

(i) if the machine in question correctly computes COMP on all inputs of length m, the test declares success with high probability, and
(ii) if the test declares success with high probability, then the machine in question gives the correct answer of COMP(y) with high probability.

Given such a test, the recovery procedure consists of iterating through each machine in the list in turn. We take the first machine P to pass testing, simulate $P(y)$ some number of times and output the majority answer. Given a testing procedure with properties (i) and (ii), correctness of this procedure follows using standard probability arguments (Chernoff and union bounds) and the assumption that we are guaranteed that at least one machine in the list of machines correctly computes COMP at length m.

The technical heart of the recovery procedure is the testing procedure to determine if a given machine P correctly computes COMP(y) for $y = \langle M, x, t, j\rangle$. This test is based on the local checkability of computation tableaux – the j^{th} bit of the configuration of $M(x)$ in time step t is determined by a constant number of bits from the configuration in time step $t-1$. For each bit (t, j) of the tableau, this gives a local consistency check – make sure that the value P claims for $\langle M, x, t, j\rangle$ is consistent with the values P claims for each of the bits of the tableau that this bit depends on. We implement this intuition as follows.

(1) For each possible t' and j', simulate $P(\langle M, x, t', j'\rangle)$ a large number of times and fail the test if the acceptance ratio lies in the range $[3/8, 5/8]$.
(2) For each possible t' and j', do the following. Let j'_1, ..., j'_k be the bits of the configuration in time step $t'-1$ that bit j' in time step t' depends on. Simulate each of $P(\langle M, x, t', j'\rangle)$, $P(\langle M, x, t'-1, j'_1\rangle)$, ..., $P(\langle M, x, t'-1, j'_k\rangle)$ a large number of times. If the majority values of these simulations are not consistent with the transition function of M, then fail the test. For example, if the bit in column j' should not change from time $t'-1$ to time t', but P has claimed different values for these bits, fail the test.

Each time we need to run multiple trials of P, we run $2^{O(s(m))}$ many. The first test checks that P has error bounded away from $1/2$ on input $\langle M, x, t, j\rangle$ and on all other bits of the computation tableau of $M(x)$. This allows us to amplify the error probability of P to exponentially small in $2^{s(m)}$. For some constants $0 < \gamma < \delta < 1/2$, the first test has the following properties: (A) If P passes the test with non-negligible probability then for any t' and j', the random variable $P(\langle M, x, t', j'\rangle)$ deviates from its majority value with probability less than δ, and (B) if the latter is the case with δ replaced by γ then P passes the test with overwhelming probability. The second test verifies the local consistency of the computation tableau claimed by P. Note that if P computes COMP correctly at length m then P passes each consistency test with high probability, and if P passes each consistency test with high probability then P must compute the correct value for COMP(y). This along with the two properties of the first test guarantee that we can choose a large enough number of trials for the second test so that properties (i) and (ii) from above are satisfied.

Consider the space usage of the recovery procedure. The main tasks are the following: (a) cycle over all machines in the list of machines, and (b) for each t' and j' determine the

bits of the tableau that bit (t', j') depends on and for each of these run $2^{O(s(m))}$ simulations of P. The first requirement depends on the representation of the list of machines. For our application, we will be cycling over all advice strings for input length m, and this takes $O(s(m))$ space provided advice strings for M_i are of length at most $s(m)$. The second requirement takes an additional $O(s(m))$ space by the fact that we only need to simulate P while it uses $s(m)$ space and the fact that the computation tableau bits that bit (t', j') depends on are constantly many and can be computed very efficiently.

5. Zero-sided error Recovery Procedure – Configuration Reachability

In this section we develop a space-efficient recovery procedure for the configuration reachability language (hereafter written CONFIG), the hard language used in the construction of Theorem 1.2.

> CONFIG = $\{\langle M, x, c_1, c_2, t\rangle \,|\, M$ is a nondeterministic Turing machine, and on input x, if M is in configuration c_1, then configuration c_2 is reachable within t time steps.

We point out that CONFIG is "hard" for one-sided error machines since a one-sided error machine can also be viewed as a nondeterministic machine. That is, if we want to know whether $\Pr[M_i(x) = 1] < \frac{1}{2}$ for M_i a one-sided error machine that uses $s(n)$ space, we can query the CONFIG instance $\left\langle M_i, x, c_1, c_2, 2^{O(s(|x|))}\right\rangle$ where c_1 is the unique start configuration, and c_2 is the unique accepting configuration.

We now present the recovery procedure for CONFIG. We wish to compute CONFIG on inputs of length m with *zero-sided error* and in space $O(s(m))$ when given a list of randomized machines with the guarantee that at least one of the machines computes CONFIG on all inputs of length m using $s(m)$ space with *one-sided error*. Let $y = \langle M, x, c_1, c_2, t\rangle$ be an instance of CONFIG with $|y| = m$ that we wish to compute.

As we need to compute CONFIG with zero-sided error, we can only output a value of "yes" or "no" if we are sure this is correct. The outer loop of our recovery procedure is the following: cycle through each machine in the list of machines, and for each execute a search procedure that attempts to verify whether configuration c_2 is reachable from configuration c_1. The search procedure may output "yes", "no", or "fail", and should have the following properties:

(i) if the machine in question correctly computes CONFIG at length m, the search procedure comes to a definite answer ("yes" or "no") with high probability, and
(ii) when the search procedure comes to a definite answer, it is always correct, no matter what the behavior of the machine in question.

We cycle through all machines in the list, and if the search procedure ever outputs "yes" or "no", we halt and output that response. If the search procedure fails for all machines in the list, we output "fail". Given a search procedure with properties (i) and (ii), the correctness of the recovery procedure follows from the fact that we are guaranteed that one of the machines in the list of machines correctly computes CONFIG at length m.

The technical heart of the recovery procedure is a search procedure with properties (i) and (ii). Let P be a randomized machine under consideration, and $y = \langle M, x, c_1, c_2, t\rangle$ an input of length m we wish to compute. Briefly, the main idea is to mimic the proof that NL=coNL to verify reachability and un-reachability, replacing nondeterministic guesses with simulations of P. If P computes CONFIG at length m correctly, there is a high

probability that we have correct answers to all nondeterministic guesses, meaning property (i) is satisfied. Property (ii) follows from the fact that the algorithm can discover when incorrect nondeterministic guesses have been made. For completeness, we explain how the nondeterministic algorithm of [8, 14] is used in our setting. The search procedure works as follows.

(1) Let k_0 be the number of configurations reachable from c_1 within 0 steps, i.e., $k_0 = 1$.
(2) For each value $\ell = 1, 2, ..., t$, compute the number k_ℓ of configurations reachable within ℓ steps of c_1, using only the fact that we have remembered the value $k_{\ell-1}$ that was computed in the previous iteration.
(3) While computing k_t, experience all of these configurations to see if c_2 is among them.

Consider the portion of the second step where we must compute k_ℓ given that we have already computed $k_{\ell-1}$. We accomplish this by cycling through all configurations c and for each one re-experiencing all configurations reachable from c_1 within $\ell-1$ steps and verifying whether c can be reached in at most one step from at least one of them. To re-experience configurations reachable within distance $\ell - 1$, we try all possible configurations and query P to verify a nondeterministic path to each. To check if c is reachable within one step of a given configuration, we use the transition function of M. If we fail to re-experience all $k_{\ell-1}$ configurations or if P gives information inconsistent with the transition function of M at any point we consider the search for reachability/un-reachability failed with machine P.

An examination of the algorithm reveals that it has property (ii) from above: if the procedure reaches a "yes" or "no" conclusion for reachability, it must be correct. Further, by using a large enough number of trials each time we simulate P, we can ensure that we get correct answers on every simulation of P with high probability if P correctly computes CONFIG at length m. This implies property (i) from above.

Consider the space usage of the recovery procedure. A critical component is to be able to cycle over all configurations and determine whether two configurations are "adjacent". As the instances of CONFIG we are interested in correspond to a machine which uses $s(n)$ space, these two tasks can be accomplished in $O(s(m))$ space. The remaining tasks of the recovery procedure take $O(s(m))$ space for similar reasons as given for the recovery procedure for the computation tableau language in the previous section.

6. Analysis

In this section we explain how we come to the parameters given in the statements of Theorems 1.1 and 1.2. First, consider the space usage of the construction. The recovery procedures use $O(s(m))$ space when dealing with inputs of size m, and the additional tasks of the diagonalizing machine N also take $O(s(m))$ space. For input lengths n where N is responsible for copying down the delayed computation of the hard language L, N executes the recovery procedure using M_i on padded inputs of one larger length. Thus for such input lengths, the space usage of N is $O(s(n + 1))$. For input length n_i, N produces an instance y of the hard language corresponding to complementary behavior of M_i on inputs of length n_i and then simulates $N(y)$. For two-sided error machines, we reduce to the computation tableau language COMP. When M_i is allowed $s(n)$ space, the resulting instance of COMP is of size $n+O(s(n))$. For one- and zero-sided error machines, we reduce to configuration reachability, and the resulting instance is also of size $n + O(s(n))$. In both cases, the space usage of N on inputs of length n_i is $O(s(n_i + O(s(n_i))))$. We have chosen COMP and CONFIG as hard languages over other natural candidates (such as the circuit

value problem for Theorem 1.1 and st-connectivity for Theorem 1.2) because COMP and CONFIG minimize the blowup in input size incurred by using the reductions.

The constant hidden in the big-O notation depends on things such as the alphabet size of M_i. If $s'(n) = \omega(s(n + as(n)))$ for all constants a, N operating in space $s'(n)$ has enough space to diagonalize against each M_i for large enough n. To ensure the asymptotic behavior has taken effect, we have N perform the construction against each machine M_i infinitely often. We set N's advice bit to zero on the entire interval of input lengths if N does not yet have sufficient space. Note that this use of advice obviates the need for $s'(n)$ to be space constructible.

Finally consider the amount of advice that the smaller space machines can be given. As long as the advice is at most $s(n)$, the recovery procedure can efficiently cycle through all candidate machines (M_i with each possible advice string). Also, to complement M_i for each advice string at length n_i, we need at least one input for each advice string of length n_i. Thus, the amount of advice that can be allowed is $\min(s(n), n)$.

Acknowledgments

We thank Scott Diehl for many useful discussions, in particular pertaining to the proof of Theorem 1.2. We also thank the anonymous reviewers for their time and suggestions.

References

[1] B. Barak. A probabilistic-time hierarchy theorem for slightly non-uniform algorithms. In *Workshop on Randomization and Approximation Techniques in Computer Science*, 2002.

[2] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In *Fundamentals of Computation Theory*, pp. 168–179, 1991.

[3] A. Condon. The complexity of space bounded interactive proof systems. In S. Homer, U. Schöning, K. Ambos-Spies, eds, *Complexity Theory: Current Research*, pp. 147–190. Cambridge U. Press, 1993.

[4] S. Cook. A hierarchy theorem for nondeterministic time complexity. *J. Comp. Syst. Sciences*, 7:343–353, 1973.

[5] L. Fortnow and R. Santhanam. Hierarchy theorems for probabilistic polynomial time. In *IEEE Symposium on Foundations of Computer Science*, pp. 316–324, 2004.

[6] L. Fortnow, R. Santhanam, and L. Trevisan. Hierarchies for semantic classes. In *ACM Symposium on the Theory of Computing*, pp. 348–355, 2005.

[7] O. Goldreich, M. Sudan, and L. Trevisan. From logarithmic advice to singe-bit advice. Technical Report TR-04-093, Electronic Colloquium on Computational Complexity, 2004.

[8] N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Computing*, 17(5):935–938, 1988.

[9] R. Karp and R. Lipton. Turing machines that take advice. *L'Enseign. Mathém.*, 28(2):191–209, 1982.

[10] M. Karpinski and R. Verbeek. Randomness, provability, and the separation of Monte Carlo time and space. In *Computation Theory and Logic*, pp. 189–207, 1987.

[11] D. van Melkebeek and K. Pervyshev. A generic time hierarchy for semantic models with one bit of advice. *Computational Complexity*, 16:139–179, 2007.

[12] M. Saks and S. Zhou. $BP_HSPACE(S) \subseteq DSPACE(S^{3/2})$. *J. Comp. Syst. Sciences*, 58:376–403, 1999.

[13] J. Seiferas, M. Fischer, and A. Meyer. Separating nondeterministic time complexity classes. *J. ACM*, 25:146–167, 1978.

[14] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[15] J. Watrous. On the complexity of simulating space-bounded quantum computations. *Computational Complexity*, 12:48–84, 2003.

[16] S. Žàk. A Turing machine time hierarchy. *Theoretical Computer Science*, 26:327–333, 1983.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 445-456
www.stacs-conf.org

EHRENFEUCHT-FRAÏSSÉ GOES AUTOMATIC FOR REAL ADDITION

FELIX KLAEDTKE

ETH Zurich, Department of Computer Science, Switzerland
E-mail address: felixkl@inf.ethz.ch

Abstract. Various logical theories can be decided by automata-theoretic methods. Notable examples are Presburger arithmetic $\mathrm{FO}(\mathbb{Z}, +, <)$ and the linear arithmetic over the reals $\mathrm{FO}(\mathbb{R}, +, <)$, for which effective decision procedures can be built using automata. Despite the practical use of automata to decide logical theories, many research questions are still only partly answered in this area. One of these questions is the complexity of such decision procedures and the related question about the minimal size of the automata of the languages that can be described by formulas in the respective logic. In this paper, we establish a double exponential upper bound on the automata size for $\mathrm{FO}(\mathbb{R}, +, <)$ and an exponential upper bound for the discrete order over the integers $\mathrm{FO}(\mathbb{Z}, <)$. The proofs of these upper bounds are based on Ehrenfeucht-Fraïssé games. The application of this mathematical tool has a similar flavor as in computational complexity theory, where it can often be used to establish tight upper bounds of the decision problem for logical theories.

1. Introduction

Various logical theories admit automata-based decision procedures. The idea of using automata-theoretic methods to decide logical theories goes at least back to Büchi [7]. The elements of the domain of the logical theory are encoded by words over some alphabet in such a way that equality and the relations of the logical theory correspond to regular languages. In order to decide whether a formula is satisfiable, one constructs an automaton that precisely accepts the representatives of the elements that satisfy the formula. This automaton can be constructed by recursion over the formula structure, where standard automata constructions handle the boolean connectives and quantifiers. The satisfiability problem is thus reduced to the emptiness problem for automata.

The logical theories that admit such automata-based decision procedures are often called automatic and they have been systematically studied, e.g., in [4, 12, 13]. Prominent and practically relevant examples are the weak monadic second-order theory of one successor WS1S, Presburger arithmetic $\mathrm{FO}(\mathbb{Z}, +, <)$, and the linear arithmetic over the reals $\mathrm{FO}(\mathbb{R}, +, <)$, see, e.g., [5–7]. Tools like MONA [15] and LIRA [3], which have been applied to various verification problems, implement such automata-based decision procedures for

1998 ACM Subject Classification: F.1.1,F.4.1.

Key words and phrases: automata theory, automata-based decision procedures for logical theories, upper bounds, minimal sizes of automata, linear arithmetic over the reals, first-order equivalence, complexity.

This work was supported by the Swiss National Science Foundation (SNF).

logical theories such as WS1S, Presburger arithmetic, and the linear arithmetic over the reals. Furthermore, model checkers for counter systems like FAST [1, 2] use an automata-based representation of sets definable in Presburger arithmetic.

A crude complexity analysis of an automata-based decision procedure leads to a non-elementary worst-case complexity. Namely, for every quantifier alternation there is a potential exponential blow-up in the state space of the automaton. For WS1S, this wost-case scenario actually exists, since the decision problem for WS1S has a non-elementary worst-case complexity [20,23]. However, for many other automatic logical theories, the non-elementary complexity upper bounds of automata-based decision procedures often contrasts with the known computational complexity upper bounds on the decision problems for the logical theories. Moreover, such exponential blow-ups in the state spaces of the automata are rarely observed in practice in automata-based decision procedures for Presburger arithmetic and the linear arithmetic over the reals. In fact, in many cases, one obtains a smaller automaton after eliminating a quantifier. However, only partial answers exist that explain this phenomenon.

In [14], it is shown that the size of the minimal deterministic automaton that represents a Presburger definable set is triply exponentially bounded with respect to the formula length. This upper bound is established by comparing the automata for Presburger arithmetic formulas with the formulas produced by Reddy and Loveland's quantifier-elimination method for Presburger arithmetic [22]. The proof on the upper bound in [14] is rather tedious in the sense that several auxiliary upper bounds on the formulas that are generated by the quantifier-elimination method need to be established. These additional upper bounds depend on Reddy and Loveland's quantifier-elimination method. With the slightly different quantifier-elimination method by Cooper [8], we obtain an upper bound on the automata size that has at least one additional exponent.

For the linear arithmetic over the reals, the approach of using quantifier-elimination methods to establish upper bounds on the automata sizes does not lead to a satisfactory result: an application of this approach establishes only a triple exponential upper bound on the automata size when using the quantifier-elimination method for the linear arithmetic over the reals described in [10]. The author is not aware of any quantifier-elimination method for the linear arithmetic over the reals that would lead to a upper bound on the automata size that is smaller than triple exponential. However, since there are decision procedures for the linear arithmetic over the reals that run in double exponential deterministic time [10], one might conjecture that the automata size is also doubly exponentially bounded.

The main result of this paper proves this conjecture. The presented proof of the double exponential upper bound is based on Ehrenfeucht-Fraïssé games (EF-games, for short from now on). It relates the states of a minimal automaton for a formula and the equivalence classes of a refinement of the equivalence relation determined by EF-games played over $(\mathbb{R}, +, <)$. This proof technique can also be used for other automatic logical theories to establish tight upper bounds on the automata sizes. As another example, we establish an exponential upper bound on the automata size for $\mathrm{FO}(\mathbb{Z}, <)$. Note that the best known deterministic algorithms that decide $\mathrm{FO}(\mathbb{Z}, <)$ run in exponential time [11]. In summary, the results presented in this paper shed light on the complexity of automata-based decision procedures for logical theories by identifying a relationship to EF-games.

It is worth pointing out that EF-games have already been used in similar contexts. Closely related to our work is Ladner's work [17]. He uses EF-games to show decidability of monadic second-order theories of one successor and first fragments of it. Similar to

this paper, he relates the equivalence classes determined by EF-games to automata states. However, Ladner does not focus on the automata sizes and he does not consider FO$(\mathbb{R}, +, <)$.

The use of EF-games in computational complexity theory [11] and constraint databases [21] is reminiscent of their use in this paper by partitioning the domain and connecting such a partition to the definable sets. Roughly speaking, the use EF-games for establish upper bounds on the decision problem for logical theories is as follows: The key ingredient for obtaining an upper bound for the respective logical theory is to show that the quantifiers, which can range over an infinite domain, can be relativized to a finite subset. Usually, one uses EF-games here to establish upper bounds on the sizes of such sets by analyzing the information that the formulas of a certain quantifier depth can convey. Given such a result on relativizing the quantifiers, satisfiability of a formula can be checked by an exhaustive search. The upper bounds on the sizes of the sets over which the relativized quantifiers range in turn yield upper bounds on the time and space that is needed to perform this search. For several logical theories, this use of EF-games yield tight upper bounds on the computational complexity for their decision problem.

The remainder of the paper is organized as follows. In §2, we give preliminaries. In §3, we illustrate our method by analyzing the languages that are FO$(\mathbb{Z}, <)$-definable. In §4, we analyze the languages that are FO$(\mathbb{R}, +, <)$-definable and establish the double exponential upper bound on the automata size. Finally, in §5, we draw conclusions. Due to space restrictions some proofs are omitted or sketched. They can be found in the full version of the paper, which is available from the author's web-page.

2. Preliminaries

We assume that the reader is familiar with first-order logic and automata theory over finite and infinite words. Here, we recall the needed background in these areas and fix the notation and terminology that we use in the remainder of the text.

2.1. Words and Languages

Let Σ be an alphabet. We denote the set of all finite words over Σ by Σ^* and Σ^+ denotes the set $\Sigma^* \setminus \{\varepsilon\}$, where ε is the empty word. Σ^ω is the set of all ω-words over Σ. The *concatenation* of words is written as juxtaposition. We write $|w|$ for the *length* of $w \in \Sigma^*$. We often write a word $w \in \Sigma^*$ of length $\ell \geq 0$ as $w(0) \dots w(\ell - 1)$ and an ω-word $\alpha \in \Sigma^\omega$ as $\alpha(0)\alpha(1)\alpha(2)\dots$, where $w(i)$ and $\alpha(i)$ denote the ith letter of w and α, respectively.

For a language $L \subseteq \Sigma^*$, the Nerode relation $\sim_L \subseteq \Sigma^* \times \Sigma^*$ is defined as $u \sim_L v$ iff for all $w \in \Sigma^*$, it holds that $uw \in L \Leftrightarrow vw \in L$. Analogously, for an ω-language $L \subseteq \Sigma^\omega$, we define $\sim_L \subseteq \Sigma^* \times \Sigma^*$ as $u \sim_L v$ iff for all $\gamma \in \Sigma^\omega$, it holds that $u\gamma \in L \Leftrightarrow v\gamma \in L$.

2.2. First-order Logic

The (first-order) formulas over a signature are defined as usual: they are built from variables $v_0, v_1, \dots$, the symbol $\approx$ for equality, the atomic formulas over the signature, the boolean connectives $\neg$ and $\vee$, and the quantifier $\exists$. In this paper, we only consider signatures that consist of relation symbols. The signature, its relation symbols, and the arities of its relation symbols are always clear from the context. We write $\varphi(x_1, \dots, x_r)$ when at most

the variables $x_1, \ldots, x_r$ occur free in the formula φ. The *quantifier depth* of a formula φ is recursively defined as

$$\mathrm{qd}(\varphi) := \begin{cases} \mathrm{qd}(\psi) & \text{if } \varphi = \neg\psi, \\ \max\{\mathrm{qd}(\psi), \mathrm{qd}(\psi')\} & \text{if } \varphi = \psi \vee \psi', \\ 1 + \mathrm{qd}(\psi) & \text{if } \varphi = \exists x \psi, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

A (first-order) structure over a signature consists of a nonempty universe U and it associates with each relation symbol in the signature a relation over U^r, where r is the arity of the relation symbol. We use $\mathfrak{R}$ and $\mathfrak{Z}$ to denote the structures $(\mathbb{R}, +, <)$ and $(\mathbb{Z}, <)$, respectively, where $+$ is the ternary addition relation and $<$ is the ordering relation over the reals or the integers, respectively.

Let $\mathfrak{A}$ be a structure over some signature and with the universe A. For $a_1, \ldots, a_r \in A$ and a formula $\varphi(x_1, \ldots, x_r)$, we write $\mathfrak{A} \models \varphi[a_1 \ldots, a_r]$ if φ is satisfied in $\mathfrak{A}$ when the variable x_i is interpreted as a_i, for all $1 \leq i \leq r$. For the sake of brevity, we often write $\bar{x}$ and $\bar{a}$ instead of $x_1, \ldots, x_r$ and $a_1, \ldots, a_r$, respectively.

Let $m, r \in \mathbb{N}$, $\bar{a} \in A^r$ and $\bar{b} \in A^r$. We write $\bar{a} \equiv^r_m \bar{b}$ if for all formulas $\varphi(x_1, \ldots, x_r)$ with $\mathrm{qd}(\varphi) \leq m$, it holds that $\mathfrak{A} \models \varphi[\bar{a}] \Leftrightarrow \mathfrak{A} \models \varphi[\bar{b}]$. Note that the relation $\equiv^r_m$ partitions the elements of A^r. The equivalence classes of $\equiv^r_m$ can be game-theoretically characterized by so-called Ehrenfeucht-Fraïssé games. For details on these games, see, for instance, [9]. Instead of working directly with $\equiv^r_m$, we work with refinements of it, since the reasoning about a well-chosen refinement of $\equiv^r_m$ simplifies matters. In particular, it might be difficult for $\equiv^r_m$ to directly establish an upper bound on the index of $\equiv^r_m$, to identify elements $\bar{a}, \bar{b} \in A^r$ that are in the same equivalence class, and to find a representative of an equivalence class.

2.3. Representation of Sets Definable in Real Addition

Boigelot, Jodogne, and Wolper have shown in [5] that every first-order definable set $X \subseteq \mathbb{R}^r$ in $\mathfrak{R}$ determines an ω-language L that is in the Borel class $F_\sigma \cap G_\delta$. In other words, L can be accepted by a so-called weak deterministic Büchi automaton. In fact, Boigelot, Jodogne, and Wolper have established in [5] a stronger result. First, they have proved the result for an extension of $\mathfrak{R}$ with the additional predicate $\mathbb{Z}$. Second, for a formula $\varphi(x_1, \ldots, x_r)$ over this extended structure, they have shown how to effectively construct a weak deterministic Büchi automaton that represents the set $\{\bar{a} \in \mathbb{R}^r : \mathfrak{R} \models \varphi[\bar{a}]\}$.

We recall the representation of subsets of $\mathbb{R}^r$ by ω-languages from [5]. In the remainder of the text, let $\varrho > 1$ and $\Sigma := \{0, \ldots, \varrho - 1\}$ be fixed. ϱ is called the *base*. Let $r \geq 1$.

(a) V_r denotes the set of all ω-words over the alphabet $\Sigma^r \cup \{\star\}$ of the form $v \star \gamma$, where $v \in (\Sigma^r)^+$ and $\gamma \in (\Sigma^r)^\omega$.

(b) Let $v \star \gamma$ be an ω-word in V_r with $v(0) = (v_1, \ldots, v_r)$. The ω-word $v \star \gamma$ represents the vector of real numbers with r components

$$\langle v \star \gamma \rangle := -\varrho^{|v|-1} \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_r \end{pmatrix} + \textstyle\sum_{0<i<|v|} \varrho^{|v|-i-1} \cdot v(i) + \sum_{i \geq 0} \varrho^{-i-1} \cdot \gamma(i)\,,$$

where $b_i := \left\lceil \frac{v_i}{\varrho} \right\rceil$, for $1 \leq i \leq r$. Observe that $b_i = 0$ if $v_i = 0$, and $b_i = 1$, otherwise. Here, scalar multiplication is as usual and vector addition is componentwise. Note that we do not distinguish between vectors and tuples.

(c) For a formula $\varphi(x_1, \ldots, x_r)$, we define $L(\varphi) := \{\alpha \in \mathsf{V}_r : \mathfrak{R} \models \varphi[\langle \alpha \rangle]\}$.

Note that the encoding $v \star \gamma \in \mathsf{V}_1$ of a real number is based on the ϱ's complement representation. The symbol $\star$ plays the role of a decimal point, separating the integer part v from the fractional part γ. Furthermore, the first letter determines whether a "track" represents a number that is greater than or equal to 0, or a number that is less than or equal to 0. Note that the ω-words $0 \star 0^\omega$ and $(\varrho - 1) \star (\varrho - 1)^\omega$ both represent the number 0, where b^ω denotes the infinite repetition of the letter $b \in \Sigma$.

We overload the notation $\langle \cdot \rangle$ by using it also for finite nonempty prefixes in V_r. For $v \in (\Sigma^r)^+$ and $v' \in (\Sigma^r)^*$, we write $\langle v \rangle$ and $\langle v \star v' \rangle$ for $\langle v \star \bar{0}^\omega \rangle$ and $\langle v \star v' \bar{0}^\omega \rangle$, respectively, where $\bar{0}$ denotes the vector $(0, \ldots, 0) \in \Sigma^r$.

3. Automata Upper Bound for the Ordering over the Integers

Before looking at the ω-languages that can be described by the first-order logic over $\mathfrak{R}$, we look at a simpler case. Namely, we investigate the languages that can be described by formulas over $\mathfrak{Z}$. We establish an exponential upper bound on the automata size for these languages. The purpose of investigating this simpler case first is twofold. First, it introduces the main concepts, which we also use in §4 for the ω-languages definable in the first-order logic over $\mathfrak{R}$. Second, it demonstrates the generality of the approach. The results in this section illustrate the relationship between the equivalence classes of a refinement of the equivalence relation $\equiv_m^r$ and the equivalence classes of the Nerode relation of a language described by a formula $\varphi(x_1, \ldots, x_r)$ over $\mathfrak{Z}$ with $\mathrm{qd}(\varphi) \leq m$.

Throughout this section, formulas are over $\mathfrak{Z}$'s signature, and m and r range over the natural numbers. We start with some definitions. For a formula $\varphi(x_1, \ldots, x_r)$, we define the language

$$K(\varphi) := \{v \in (\Sigma^r)^+ \,:\, \mathfrak{Z} \models \varphi[\langle v \rangle]\}\,.$$

We partition $\mathbb{Z}^r$ by the equivalence relation E_m^r that is defined as

$$\bar{a}\; E_m^r\; \bar{b} \qquad \text{iff} \qquad \begin{array}{l} \mathrm{sign}(a_i - a_j - c) = \mathrm{sign}(b_i - b_j - c), \\ \text{for all } c, i, j \in \mathbb{N} \text{ with } c \leq m \text{ and } 1 \leq i, j \leq r\,, \end{array}$$

where $\bar{a}, \bar{b} \in \mathbb{Z}^r$, and $\mathrm{sign}(x) := 0$ if $x < 0$ and $\mathrm{sign}(x) := 1$, otherwise, for $x \in \mathbb{R}$. Intuitively speaking, $\bar{a}, \bar{b} \in \mathbb{Z}^r$ are in the same equivalence class of E_m^r if the distances between their components are equal up to the threshold m.

Before we launch into the proof of establishing an upper bound on the size of the minimal deterministic automaton for a formula $\varphi(x_1, \ldots, x_r)$, we give an outline: (i) We show that $E^r_{2^{\mathrm{qd}(\varphi)}}$ refines $\equiv^r_{\mathrm{qd}(\varphi)}$. (ii) We establish an upper bound on the index of $E^r_{2^{\mathrm{qd}(\varphi)}}$. (iii) We show that $E^r_{2^{\mathrm{qd}(\varphi)}}$ has a congruence property with respect to word concatenation. (iv) By using (i) and (iii), we show that $E^r_{2^{\mathrm{qd}(\varphi)}}$ determines an equivalence relation on $(\Sigma^r)^+$ that refines the Nerode relation $\sim_{K(\varphi)}$. Finally, from (ii) we derive an upper bound on the index of $\sim_{K(\varphi)}$. Note that the equivalence classes of $\sim_{K(\varphi)}$ can be viewed as the states of the minimal deterministic finite automaton that accepts $K(\varphi)$. The properties (i) to (iv) correspond to the Lemmas 3.1 to 3.4, respectively, which are given below.

Lemma 3.1. *The equivalence relation $E^r_{2^m}$ refines the equivalence relation $\equiv^r_m$. That means, $\bar{a}\; E^r_{2^m}\; \bar{b}$ implies $\bar{a} \equiv^r_m \bar{b}$, for all $\bar{a}, \bar{b} \in \mathbb{Z}^r$.*

To prove Lemma 3.1, we apply a standard technique from model theory. First, we show that the family $(E^s_n)_{s,n\in\mathbb{N}}$ of equivalence relations has the following property:

$$\text{If } \bar{a}\ E^r_{2^{m+1}}\ \bar{b} \text{ then for every } a' \in \mathbb{Z}, \text{ there is some } b' \in \mathbb{Z} \text{ such that } (\bar{a}, a')\ E^{r+1}_{2^m}\ (\bar{b}, b'). \quad (3.1)$$

Properties of this kind are often called back-and-forth properties in the literature. Note that $E^r_{2^{m+1}}$ is symmetric. Second, we complete the proof by an induction over m, where we use the property (3.1) in the induction step for the existential quantifier.

Lemma 3.2. *The index of E^r_m is at most $r! \cdot (m+1)^r$.*

Proof. There are at most $r!$ many possibilities to order the r elements increasingly. If in such an ordering the distance between the ith element x and the $(i+1)$st element y is greater than or equal to m, we have that $\mathrm{sign}(y - x - c) = 1$, for all $c \in \mathbb{N}$ with $c \leq m$. We obtain that the index is at most $r! \cdot (m+1)^r$. ∎

Lemma 3.3. *Let $u, v \in (\Sigma^r)^+$. If $\langle u\rangle\ E^r_m\ \langle v\rangle$ then $\langle uw\rangle\ E^r_m\ \langle vw\rangle$, for all $w \in (\Sigma^r)^*$.*

Proof. Let $n := |w|$, $\bar{a} := (a_1, \ldots, a_r) := \langle u\rangle$, $\bar{b} := (b_1, \ldots, b_r) := \langle v\rangle$, and $\bar{d} := (d_1 \ldots, d_r) := \langle \bar{0}w\rangle$. We have that $\langle uw\rangle = \varrho^n\bar{a} + \bar{d}$ and $\langle vw\rangle = \varrho^n\bar{b} + \bar{d}$. Furthermore, it holds that $d_i < \varrho^n$, for all $i \in \{1, \ldots, r\}$. Let $i, j, c \in \mathbb{N}$ with $1 \leq i, j \leq r$ and $c \leq m$. We have to show that

$$\mathrm{sign}\left(\varrho^n(a_i - a_j) + d_i - d_j - c\right) = \mathrm{sign}\left(\varrho^n(b_i - b_j) + d_i - d_j - c\right). \quad (3.2)$$

Case $a_i - a_j = 0$. We have that $\mathrm{sign}(a_i - a_j) = 1 = \mathrm{sign}(a_j - a_i)$. From the assumption $\bar{a}\ E^r_m\ \bar{b}$, it follows that $\mathrm{sign}(a_i - a_j) = \mathrm{sign}(b_i - b_j)$ and $\mathrm{sign}(a_j - a_i) = \mathrm{sign}(b_j - b_i)$, and hence, $b_i - b_j = 0$. Obviously, the equality (3.2) holds.

Case $b_i - b_j = 0$. This case is symmetric to the case $a_i - a_j = 0$ above.

Case $a_i - a_j \neq 0$ and $b_i - b_j \neq 0$. For showing (3.2), it suffices to show the equality

$$\mathrm{sign}\left(a_i - a_j + \tfrac{d_i - d_j - c}{\varrho^n}\right) \;=\; \mathrm{sign}\left(b_i - b_j + \tfrac{d_i - d_j - c}{\varrho^n}\right). \quad (3.3)$$

– If $m = 0$, we have that $c = 0$ and thus $\left|\frac{d_i - d_j - c}{\varrho^n}\right| \leq \frac{|d_i - d_j|}{\varrho^n} \leq \frac{\varrho^n - 1}{\varrho^n} < 1$. Since $a_i - a_j \neq 0$ and $b_i - b_j \neq 0$ and by the assumption $\bar{a}\ E^r_0\ \bar{b}$, we conclude that the equality (3.3) holds.
– If $m > 0$, we have that $\left|\frac{d_i - d_j - c}{\varrho^n}\right| \leq \frac{|d_i - d_j| + |c|}{\varrho^n} \leq \frac{\varrho^n - 1 + |c|}{\varrho^n} \leq \frac{m(\varrho^n - 1) + m}{\varrho^n} = m$. The equality (3.3) follows from the assumption $\bar{a}\ E^r_m\ \bar{b}$. ∎

Lemma 3.4. *Let φ be a formula with at most r free variables and with quantifier depth at most m. If $\langle u\rangle\ E^r_{2^m}\ \langle v\rangle$ then $u \sim_{K(\varphi)} v$, for all $u, v \in (\Sigma^r)^+$.*

Proof. We prove the lemma by contraposition. Assume that $u \not\sim_{K(\varphi)} v$, i.e., there is a word $w \in \Sigma^*$ such that $uw \in K(\varphi) \not\Leftrightarrow vw \in K(\varphi)$. It follows that $\langle uw\rangle \not\equiv^r_m \langle vw\rangle$. By Lemma 3.1, we conclude that $\langle uw\rangle\ E^r_{2^m}\ \langle vw\rangle$ does not hold. By Lemma 3.3, we obtain that $\langle u\rangle\ E^r_{2^m}\ \langle v\rangle$ does not hold. ∎

Theorem 3.5. *Let φ be a formula. The index of $\sim_{K(\varphi)}$ is at most $1 + 2^{n^2}$, where n is the length of the formula φ, i.e., φ consists of n symbols.*

Proof. Let r be the number of free variables of φ and $m := \mathrm{qd}(\varphi)$. Note that $n \geq r + m + 1$. Without loss of generality, we assume that $r > 0$. By Lemma 3.2, we have that the index of $E^r_{2^m}$ is at most $r! \cdot (2^m + 1)^r \leq 2^{r^2 + rm + r} \leq 2^{rn} \leq 2^{n^2}$. From Lemma 3.4, it follows that $\sim_{K(\varphi)}$ partitions $(\Sigma^r)^+$ in at most 2^{n^2} equivalence classes. Note that the empty word can be in an equivalence class that is distinct from all the others. ∎

4. Automata Upper Bound for Real Addition

In this section, we establish an upper bound on the automata size for the first-order logic over $\mathfrak{R}$. The proof has a similar structure as the proof in the previous section §3. However, it is more involved. In §4.1, we define a family $(F^s_n)_{s,n\in\mathbb{N}}$ of equivalence relations. In §4.1 and §4.2, we show that $(F^s_n)_{s,n\in\mathbb{N}}$ has similar properties as the family $(E^s_n)_{s,n\in\mathbb{N}}$ defined in §3. Namely, (1) we show that each $F^r_{2^{2^{m+2}}}$ refines $\equiv^r_m$ and (2) we establish a relationship between the equivalence classes of the congruence relations determined by the definable ω-languages and equivalences classes of refinements of the equivalence relations $(F^s_n)_{s,n\in\mathbb{N}}$. Finally, in §4.3, we derive the double exponential upper bound on the size of a minimal Büchi automaton that accepts the ω-language of a formula of the first-order logic over $\mathfrak{R}$.

In the following, formulas are always over $\mathfrak{R}$'s signature, and r and m range over the natural numbers.

4.1. Partitioning the Reals by First-order Formulas

The results, which we use later, and their presentation of this subsection are based on Chapter 22 of Kozen's book [16]. Since subtle modifications are made, we provide proofs in the full version of the paper. At the end of this subsection, we comment on these modifications and their implications.

An *integer affine function* of arity r is a function $f : \mathbb{R}^r \to \mathbb{R}$ defined by a linear polynomial with integer coefficients, i.e., there are $c_0, \ldots, c_r \in \mathbb{Z}$ such that for all $x_1, \ldots, x_r \in \mathbb{R}$, it holds that $f(x_1, \ldots, x_r) = c_0 + \sum_{1\leq i\leq r} c_i x_i$. For such a function, f^* denotes the function with $f^*(x_1, \ldots, x_r) = \sum_{1\leq i\leq r} c_i x_i$, for all $x_1, \ldots, x_r \in \mathbb{R}^r$. We define $||f|| := \max\{0, |c_1|, \ldots, |c_r|\}$. Let A^r be the set of all integer affine functions of arity r and

$$B^r_m := \left\{ f \in A^r \,:\, ||f|| \leq m \text{ and } |f(\bar{0})| \leq rm \right\}.$$

Definition 4.1. We partition $\mathbb{R}^r$ by the equivalence relation F^r_m that is defined as

$$\bar{a}\ F^r_m\ \bar{b} \qquad \text{iff} \qquad \text{for all } f \in B^r_m,\ \operatorname{sign}(f(\bar{a})) = \operatorname{sign}(f(\bar{b}))\,,$$

where $\bar{a}, \bar{b} \in \mathbb{R}^r$.

Note that F^r_m decomposes $\mathbb{R}^r$ into cells. Each such cell is described by a conjunction of linear inequations, where the absolute values of the coefficients of the inequations are bounded. Moreover, we remark that the technique that we present in the following by connecting such partitions to first-order logic and Ehrenfeucht-Fraïssé games is reminiscent of techniques in computational complexity (see [11]) and constraint databases (see [21]). A novel insight is that these partitions are also connected to the relation $\sim_{L(\varphi)}$ for a formula φ. We start with some properties about the family $(F^s_n)_{s,n\in\mathbb{N}}$ of equivalence relations.

Lemma 4.2. *Let $\bar{a}, \bar{b} \in \mathbb{R}^r$. If $\bar{a}\ F^r_{4m^2}\ \bar{b}$ then for all $a' \in \mathbb{R}$, there is some $b' \in \mathbb{R}$ such that $(\bar{a}, a')\ F^{r+1}_m\ (\bar{b}, b')$.*

Similar to Lemma 3.1, we obtain the following lemma by using Lemma 4.2.

Lemma 4.3. *For all $\bar{a}, \bar{b} \in \mathbb{R}^r$, it holds that if $\bar{a}\ F^r_{2^{2^{m+2}}}\ \bar{b}$ then $\bar{a} \equiv^r_m \bar{b}$.*

The following two lemmas show how to obtain a set $R \subseteq \mathbb{R}^r$ such that each equivalence class of F^r_m has at least one representative in R. Let σ be an equivalence class of F^r_m and let σ' be an equivalence class of F^{r+1}_n, where $n \in \mathbb{N}$. We say that σ' is *consistent* with σ if $(\sigma \times \mathbb{R}) \cap \sigma' \neq \emptyset$.

Lemma 4.4. *For each equivalence class σ of F^1_m, we have that*

$$\sigma \cap \left\{ \tfrac{d}{c} \,:\, c,d \in \mathbb{Z} \text{ with } c \neq 0 \text{ and } |c|,|d| \leq 2m^2 \right\} \neq \emptyset\,.$$

Lemma 4.5. *Let $r > 1$, $\bar{a} \in \mathbb{R}^r$, where σ is the equivalence class of $\bar{a}$ with respect to $F^r_{2m^2}$. For every equivalence class σ' of F^{r+1}_m that is consistent with σ, we have that*

$$\sigma' \cap \left\{ \left(\bar{a}, \tfrac{f(\bar{a})+d}{c}\right) \,:\, f \in B^r_2 \text{ and } c,d \in \mathbb{Z} \setminus \{0\} \text{ with } |c| \leq 2, \text{ and } |d| < 2 \right\} \neq \emptyset$$

if $m = 1$, and, for $m \neq 1$, we have that

$$\sigma' \cap \left\{ \left(\bar{a}, \tfrac{f(\bar{a})}{c}\right) \,:\, f \in B^r_{2m^2} \text{ and } c \in \mathbb{Z} \setminus \{0\} \text{ with } |c| \leq 2m^2 \right\} \neq \emptyset\,.$$

Remark 4.6. Before we proceed to establish the upper bound on the size of the minimal automata representation for the set defined by a formula φ, we point out the differences between the family $(F^s_n)_{s,n\in\mathbb{N}}$ of equivalence relations and the family of equivalence relations defined in Kozen's book [16] in Chapter 22.

In Kozen's book, two elements $\bar{a}, \bar{b} \in \mathbb{R}^r$ are related iff $\operatorname{sgn}(f(\bar{a})) = \operatorname{sgn}(f(\bar{b}))$, for all integer affine function $f \in A^r$ with $||f|| \leq m$ and $|f(\bar{0})| \leq m$. Here, sgn denotes the signum function that is defined as $\operatorname{sgn}(x) := -1$ if $x < 0$, $\operatorname{sgn}(x) := 1$ if $x > 0$, and $\operatorname{sgn}(0) := 0$.

There are two differences to our definition. First, we use the function sign instead of the function sgn. This difference is actually irrelevant. Using sign instead of sgn in the definition in Kozen's book would not change the equivalence relations. However, we found the reasoning in the proofs when using the function sign slightly simpler. Second and more relevant, we require $|f(\bar{0})| \leq rm$ instead of $|f(\bar{0})| \leq m$. The proofs of the Lemmas 4.2 to 4.5 follow the lines of the proofs of the corresponding lemmas in Kozen's book. However, there are subtle differences, e.g., in Lemma 4.5, we have the special case for $m = 1$, which is not needed in the corresponding lemma in Kozen's book.

An immediate consequence of only requiring this weaker restriction on the functions $f \in A^r$ is that the equivalence relation F^r_m refines the corresponding equivalence relation as defined in Kozen's book. The purpose for having finer equivalence relations is the following: For a formula $\varphi(x_1,\ldots,x_r)$, we show in §4.2 that the equivalence classes of $\sim_{L(\varphi)}$ are related to the equivalence classes of a certain relation in the family $(F^s_n)_{s,n\in\mathbb{N}}$. Without the weaker requirement we were not able to establish a similar relationship. The problem can be pinpointed to Lemma 4.8, which is crucial in relating the equivalence relations. The corresponding statement of Lemma 4.8 would not be correct when using the equivalence relations as defined in Kozen's book.

4.2. Relationship to Languages

In this subsection, we establish a relationship between the equivalence relation $F^r_{2^{2^{m+2}+1}}$ and the congruence relation $\sim_{L(\varphi)}$, where $\varphi(x_1,\ldots,x_r)$ is a formula with $\operatorname{qd}(\varphi) \leq m$. Namely, we show that $F^r_{2^{2^{m+2}+1}}$ determines a refinement of the congruence relation $\sim_{L(\varphi)}$.

We start with a technical lemma. Its proof is straightforward and we therefore omit it. In the following, we will use it without explicitly referring to it.

Lemma 4.7. *For $f \in A^r$, $u \in (\Sigma^r)^+$, $u' \in (\Sigma^r)^*$, and $\gamma \in (\Sigma^r)^\omega$, the following facts hold:*
(1) $f(\langle uu' \rangle) = f(\bar{0}) + \varrho^{|u'|} f^*(\langle u \rangle) + f^*(\langle \bar{0} u' \rangle)$, *and*
(2) $f(\langle u \star u' \gamma \rangle) = f(\bar{0}) + f^*(\langle u \star u' \rangle) + \varrho^{-|u'|} f^*(\langle \bar{0} \star \gamma \rangle)$.

The next two lemmas show that the equivalence relations in the family $(F_n^s)_{s,n\in\mathbb{N}}$ have congruence properties on words with respect to word concatenation and show how their equivalence classes relate to the equivalence classes of the congruence relation $\sim_{L(\varphi)}$. We want to point out a technical detail, which is reflected in the (b)-parts of the lemmas, is illustrated by the following example. The words $u\star u'$ and $u\star u'\bar{0}$ represent the same vector of real numbers, i.e., $\langle u\star u'\rangle = \langle u\star u'\bar{0}\rangle$. Therefore, $u\star u'$ and $u\star u'\bar{0}$ represent the same equivalence class in F_m^r. However, $u\star u'$ and $u\star u'\bar{0}$ might not be in the same equivalence class with respect to $\sim_{L(\varphi)}$. Observe that appending an ω-word $\gamma\in(\Sigma^r)^\omega$ to $u\star u'$ and $u\star u'\bar{0}$ may yield representations of different vectors of real numbers, i.e., $\langle u\star u'\gamma\rangle \not\equiv \langle u\star u'\bar{0}\gamma\rangle$, and $u\star u'\gamma$ and $u\star u'\bar{0}\gamma$ may represent different equivalence classes in F_m^r.

Lemma 4.8. *For all $u,v\in(\Sigma^r)^+$ and $u',v'\in(\Sigma^r)^*$, the following two facts hold:*
(a) If $\langle u\rangle\ F_m^r\ \langle v\rangle$ then for all $w\in(\Sigma^r)^*$, $\langle uw\rangle\ F_m^r\ \langle vw\rangle$.
(b) If $\langle u\star u'\rangle\ F_{2m}^r\ \langle v\star v'\rangle$ and $|u'|\geq|v'|$ then for all $\gamma\in(\Sigma^r)^\omega$, $\langle u\star u'\gamma\rangle\ F_m^r\ \langle v\star v'\bar{0}^k\gamma\rangle$ with $k=\min\{|u'|-|v'|\}\cup\{k\in\mathbb{Z}\,:\,\varrho^k\geq rm\}$.

Proof. For $r=0$, there is nothing to prove. In the following, we assume that $r>0$.
(a) We prove (a) by contraposition. Assume that for some $w\in(\Sigma^r)^*$, it is not the case that $\langle uw\rangle\ F_m^r\ \langle vw\rangle$, i.e., there is some $f\in B_m^r$ with $\operatorname{sign}(f(\langle uw\rangle))\neq\operatorname{sign}(f(\langle vw\rangle))$. Without loss of generality, we assume that $f(\langle uw\rangle)<0$ and hence $f(\langle vw\rangle)\geq 0$. The other cases can be reduced to this case by using the function $g\in B_m^r$ with $g(\bar{x})=-f(\bar{x})$, for all $\bar{x}\in\mathbb{R}^r$.

We have that $\varrho^{|w|}f^*(\langle u\rangle)+f(\langle\bar{0}w\rangle)<0$ and $\varrho^{|w|}f^*(\langle v\rangle)+f(\langle\bar{0}w\rangle)\geq 0$. Obviously, it must hold that $f^*(\langle u\rangle)\neq f^*(\langle v\rangle)$. If $\operatorname{sign}(f^*(\langle u\rangle))\neq\operatorname{sign}(f^*(\langle v\rangle))$ then $\langle u\rangle\ F_m^r\ \langle v\rangle$ does not hold and we are done. So, assume that $\operatorname{sign}(f^*(\langle u\rangle))=\operatorname{sign}(f^*(\langle v\rangle))$. If $|f^*(\langle u\rangle)|\leq rm$ or $|f^*(\langle v\rangle)|\leq rm$ then we are also done by choosing an appropriate function $g\in B_m^r$ with $\operatorname{sign}(g(\langle u\rangle))\neq\operatorname{sign}(g(\langle v\rangle))$. So, assume that $|f^*(\langle u\rangle)|,|f^*(\langle v\rangle)|>rm$. Note that $|f^*(\langle\bar{0}w\rangle)|\leq(\varrho^{|w|}-1)rm$.
– If $f^*(\langle v\rangle)<-rm$, we obtain a contradiction to the assumption $f(\langle vw\rangle)\geq 0$, since

$$\begin{aligned}\varrho^{|w|}f^*(\langle v\rangle)+f(\langle\bar{0}w\rangle) &= \varrho^{|w|}f^*(\langle v\rangle)+f^*(\langle\bar{0}w\rangle)+f(\bar{0})\\ &< -\varrho^{|w|}rm+(\varrho^{|w|}-1)rm+rm\leq 0\,.\end{aligned}$$

– If $f^*(\langle v\rangle)>rm$, we conclude that $f^*(\langle u\rangle)>rm$. Analogously, as in the above case, we obtain a contradiction to the assumption $f(\langle uw\rangle)<0$.
(b) Let f be an arbitrary function in B_m^r and $\gamma\in(\Sigma^r)^\omega$. We have to show that $\operatorname{sign}(f(\langle u\star u'\gamma\rangle))=\operatorname{sign}(f(\langle v\star v'\bar{0}^k\gamma\rangle))$. Since $B_m^r\subseteq B_{2m}^r$, it follows from the assumption $\langle u\star u'\rangle F_{2m}^r\langle v\star v'\rangle$ that $\operatorname{sign}(f(\langle u\star u'\rangle))=\operatorname{sign}(f(\langle v\star v'\rangle))$. That means, either (1) $f(\langle u\star u'\rangle),f(\langle v\star v'\rangle)<0$ or (2) $f(\langle u\star u'\rangle),f(\langle v\star v'\rangle)\geq 0$ holds. Since the case (1) can be reduced to the case (2) by considering the function $g(\bar{x})=-f(\bar{x})$, for all $\bar{x}\in\mathbb{R}^r$, we restrict ourselves to (2).

For the sake of readability, we use the abbreviations $a:=f^*(\langle u\star u'\rangle)$, $b:=f^*(\langle v\star v'\rangle)$, and $c:=f^*(\langle\bar{0}\star\gamma\rangle)$. Note that

$$f(\langle u\star u'\gamma\rangle)=f(\bar{0})+a+c\varrho^{-|u'|}\qquad\text{and}\qquad f(\langle v\star v'\bar{0}^k\gamma\rangle)=f(\bar{0})+b+c\varrho^{-|v'|-k}\,.\quad(4.1)$$

If $c\geq 0$ then $\operatorname{sign}(f(\langle u\star u'\gamma\rangle))=\operatorname{sign}(f(\langle v\star v'\bar{0}^k\gamma\rangle))=1$. In the following, assume $c<0$.

Case $a\neq b$. With the assumption $\langle u\star u'\rangle F_{2m}^r\langle v\star v'\rangle$ we conclude that $a,b>2rm$. Note that $|f^*(\langle\bar{0}\star\alpha\rangle)|\leq rm$, for all $\alpha\in(\Sigma^r)^\omega$. It follows that

$$f^*(\langle u\star u'\gamma\rangle)=a+c\varrho^{-|u'|}>2rm-rm\geq rm\,.$$

The reasoning for $f^*(\langle v \star v' \bar{0}^k \gamma \rangle) > rm$ is similar. Since $|f(\bar{0})| \leq rm$, we have that $\mathrm{sign}(f(\langle u \star u' \gamma \rangle)) = \mathrm{sign}(f(\langle v \star v' \bar{0}^k \gamma \rangle)) = 1$.

Case $a = b$. For $k = |u'| - |v'|$, it immediately follows from the equalities in (4.1) that $f(\langle u \star u' \gamma \rangle) = f(\langle v \star v' \bar{0}^k \gamma \rangle)$, and hence $\mathrm{sign}(f(\langle u \star u' \gamma \rangle)) = \mathrm{sign}(f(\langle v \star v' \bar{0}^k \gamma \rangle))$. For $a = b = -f(\bar{0})$, it is also straightforward to see from the two equalities in (4.1) that $\mathrm{sign}(f(\langle u \star u' \gamma \rangle)) = \mathrm{sign}(f(\langle v \star v' \bar{0}^k \gamma \rangle))$. For the rest of the proof, assume $k = \min\{k \in \mathbb{Z} : \varrho^k \geq rm\}$ and $b \neq -f(\bar{0})$. Moreover, for $|c| \cdot \varrho^{-|u'|} > f(\bar{0}) + a$, it follows directly from the equalities (4.1) that $\mathrm{sign}(f(\langle u \star u' \gamma \rangle)) = \mathrm{sign}(f(\langle v \star v' \bar{0}^k \gamma \rangle)) = 0$. So, we also assume that $|c| \cdot \varrho^{-|u'|} \leq f(\bar{0}) + a$. Observe that $f(\langle u \star u' \gamma \rangle) \geq 0$. Furthermore, observe that $f(\bar{0}) + b \geq \varrho^{-|v'|}$. We have that $f(\langle v \star v' \bar{0}^k \gamma \rangle) \geq \frac{1}{\varrho^{|v'|}} + \frac{c}{\varrho^{|v'|+k}} = \frac{\varrho^k - |c|}{\varrho^{|v'|+k}} \geq \frac{rm - rm}{\varrho^{|v'|+k}} \geq 0$. ∎

Lemma 4.9. *Let $\varphi(x_1, \ldots, x_r)$ be a formula with $\mathrm{qd}(\varphi) \leq m$. For all $u, v \in (\Sigma^r)^+$ and $u', v' \in (\Sigma^r)^*$, the following two facts hold:*
(a) If $\langle u \rangle \; F^r_{2^{2^{m+2}+1}} \; \langle v \rangle$ then $u \sim_{L(\varphi)} v$.
(b) If $\langle u \star u' \rangle \; F^r_{2^{2^{m+2}+1}} \; \langle v \star v' \rangle$ and $|u'| \geq |v'|$ then $u \star u' \sim_{L(\varphi)} v \star v' \bar{0}^k$ with $k = \min\{|u'| - |v'|\} \cup \{k \in \mathbb{Z} : \varrho^k \geq rm\}$.

Proof. We only show (a). The proof for (b) is analogous and we omit it. From Lemma 4.8(a), it follows that $\langle uw \rangle \; F^r_{2^{2^{m+2}+1}} \; \langle vw \rangle$, for all $w \in (\Sigma^r)^*$. With Lemma 4.8(b), we obtain that $\langle uw \star \gamma \rangle \; F^r_{2^{2^{m+2}}} \; \langle vw \star \gamma \rangle$, for all $w \in (\Sigma^r)^*$ and $\gamma \in (\Sigma^r)^\omega$. By Lemma 4.3, we conclude that $\langle uw \star \gamma \rangle \equiv^r_m \langle vw \star \gamma \rangle$, for all $w \in (\Sigma^r)^*$ and $\gamma \in (\Sigma^r)^\omega$. In particular, we have that $uw \star \gamma \in L(\varphi) \Leftrightarrow vw \star \gamma \in L(\varphi)$, for all $w \in (\Sigma^r)^*$ and $\gamma \in (\Sigma^r)^\omega$. From this it follows that $u \sim_{L(\varphi)} v$, since for any ω-word α not in V_r, we have that $u\alpha, v\alpha \notin L(\varphi)$. ∎

4.3. Upper Bounds

We establish an upper bound on the index of F^r_m, from which we then derive an upper bound on the automata size. We start with a simple lemma.

Lemma 4.10. *The cardinality of B^r_m is at most $(2rm + 1)(2m + 1)^r$.*

Using the Lemmas 4.4, 4.5, and 4.10, we establish an upper bound on the index of F^r_m.

Lemma 4.11. *The index of F^r_m is at most $\max\{1, m^{2^{3+r}} \cdot 2^{2^{3+r}}\}$.*

Theorem 4.12. *Let φ be a formula. The index of $\sim_{L(\varphi)}$ is at most $2^{2^{8+n}}$, where n is the length of the formula φ, i.e., the number of symbols of φ.*

Proof. Let r be the number of free variables in φ and $m := \mathrm{qd}(\varphi)$. We use $F^r_{2^{2^{m+2}+1}}$ to define a refinement R of $\sim_{L(\varphi)}$. First, the singleton $\{\varepsilon\}$ is an equivalence class of R. Second, the set of words with at least two occurrences of the letter $\star$ is another equivalence class of R. The equivalence class of a word $v \in (\Sigma^r)^+$ of R is $\{u \in (\Sigma^r)^+ : \langle v \rangle \; F^r_{2^{2^{m+2}+1}} \; \langle u \rangle\}$.

It remains to define the equivalence classes of R on $F := \{v \star v' : v \in (\Sigma^r)^+ \text{ and } v' \in (\Sigma^r)^*\}$. For $v \star v' \in F$, let $S := \{u \star u' \in F : \langle v \star v' \rangle \; F^r_{2^{2^{m+2}+1}} \; \langle u \star u' \rangle\}$. R chops S into equivalence classes, assuming $|v'| \leq |u'|$, for all $u \star u' \in S$:
– For $k \in \{0, \ldots, \lceil \log_\varrho r 2^{2^{m+2}+1} \rceil - 1\}$, the equivalence class of $v \star v' \bar{0}^k$ of R is $\{u \star u' \in S : |u'| = |v'| + k\}$.
– For $k = \lceil \log_\varrho r 2^{2^{m+2}+1} \rceil$, the equivalence class of $v \star v' \bar{0}^k$ of R is $\{u \star u' \in S : |u'| \geq |v'| + k\}$.

Note that any word $u \star u' \in S$ relates to exactly one word $v \star v'\bar{0}^k$.

With Lemma 4.9 at hand, it is easy to see that R refines $\sim_{L(\varphi)}$. It remains to prove an upper bound on the index of R. Note that $n \geq m + r \geq 1$. By Lemma 4.11, an upper bound on the index of $F^r_{2^{2^{m+2}+1}}$ is

$$\left(2^{2^{m+2}+1}\right)^{2^{3+r}} \cdot 2^{2^{3+r}} = 2^{2^{m+2} \cdot 2^{3+r} + 2^{3+r} + 2^{3+r}} = 2^{2^{5+r+m} + 2^{4+r}} \leq 2^{2^{6+n}} .$$

Hence, R partitions $(\Sigma^r)^+$ into at most $2^{2^{6+n}}$ equivalence classes and F is partitioned into at most $2^{2^{6+n}} \cdot \lceil \log_\varrho r 2^{2^{m+2}+1} \rceil \geq 2^{2^{6+n}} \cdot r(2^{m+2}+1) \geq 2^{2^{6+n}} \cdot 2^{3+n} \geq 2^{2^{7+n}}$ equivalence classes. From this, we derive the upper bound $2^{2^{8+n}}$ on R's index. ∎

Remark 4.13. Since for any formula φ, $L(\varphi)$ is an ω-language in the Borel class $F_\sigma \cap G_\delta$ [5], we can—similar to deterministic finite automata—view the equivalence classes of $\sim_{L(\varphi)}$ as the states of a minimal deterministic Büchi automaton that accepts $L(\varphi)$. For further details, see [19] and [18]. Thus, Theorem 4.12 establishes a double exponential upper bound with respect to the formula length on the size of the minimal number of states of any Büchi automaton that accepts $L(\varphi)$.

Remark 4.14. The double exponential upper bound on the automata size is tight, i.e., there is a family of formulas $(\varphi_n)_{n \in \mathbb{N}}$ such that for each $n \in \mathbb{N}$, the length of φ_n is linear in n and the index of $\sim_{L(\varphi_n)}$ is double exponential in n. An analogous result with a similar proof has already been shown in [14] for Presburger arithmetic.

5. Conclusion

This papers presented a new method to reason about the sizes of automata that represent first-order definable sets of automatic structures. The method consists of identifying a relationship between the states of a minimal deterministic automaton for a formula and the equivalence classes of a refinement of the equivalence relation determined by Ehrenfeucht-Fraïssé games. We applied the presented method to establish tight upper bounds on the minimal sizes of automata that represent sets definable in $\mathrm{FO}(\mathbb{Z}, <)$ and $\mathrm{FO}(\mathbb{R}, +, <)$. For $\mathrm{FO}(\mathbb{R}, +, <)$, previously proposed techniques based on quantifier-elimination methods [14] failed to establish a double exponential upper bound on the automata size.

As future work, we want to investigate how, and to what extent, the upper bounds on the automata sizes depend on how elements of a structure are encoded as words. The word encoding of integers and reals that we have used in this paper is based on the ϱ's complement representation, for some $\varrho \in \mathbb{N}$ with $\varrho \geq 2$. There are various other word encodings of numbers so that, e.g., $\mathrm{FO}(\mathbb{Z}, <)$ admits an automata-based decision procedure. For a study on the impact of encodings in automatic structures, see, e.g., [13]. We also plan to apply the presented technique to establish further upper bounds on automata sizes for other automatic structures and use it to simplify the proofs of previously established upper bounds. For instance, for Presburger arithmetic, we expect that we can use equivalence relations similar to the ones used in this paper for $\mathrm{FO}(\mathbb{R}, +, <)$. However, we have to adjust the bounds on the coefficients and take the definable divisibility relations into account.

Acknowledgments. The author thanks David Basin, Cas Cremers, Matthias Schmalz, and the anonymous reviewers for their comments on earlier versions of the paper.

References

[1] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci, *FAST: Fast acceleration of symbolic transition systems*, in Proc. of the 15th Int. Conf. on Computer Aided Verification (CAV), vol. 2725 of Lect. Notes Comput. Sci., Springer, 2003, pp. 118–121.

[2] S. Bardin, J. Leroux, and G. Point, *FAST extended release*, in Proc. of the 18th Int. Conf. on Computer Aided Verification (CAV), vol. 4144 of Lect. Notes Comput. Sci., Springer, 2006, pp. 63–66.

[3] B. Becker, C. Dax, J. Eisinger, and F. Klaedtke, *LIRA: Handling constraints of linear arithmetics over the integers and the reals*, in Proc. of the 19th Int. Conf. on Computer Aided Verification (CAV), vol. 4590 of Lect. Notes Comput. Sci., Springer, 2007, pp. 307–310.

[4] A. Blumensath and E. Grädel, *Finite presentations of infinite structures: Automata and interpretations*, Theory Comput. Syst., 37 (2004), pp. 641–674.

[5] B. Boigelot, S. Jodogne, and P. Wolper, *An effective decision procedure for linear arithmetic over the integers and reals*, ACM Trans. Comput. Log., 6 (2005), pp. 614–633.

[6] B. Boigelot and P. Wolper, *Representing arithmetic constraints with finite automata: an overview*, in Proc. of the 18th Int. Conf. on Logic Programming (ICLP), vol. 2401 of Lect. Notes Comput. Sci., Springer, 2002, pp. 1–19.

[7] J. Büchi, *Weak second-order arithmetic and finite automata*, Z. Math. Logik Grundlagen Math., 6 (1960), pp. 66–92.

[8] D. C. Cooper, *Theorem proving in arithmetic without multiplication*, in Proc. 7th Annual Machine Intelligence Workshop, B. Meltzer and D. Michie, eds., Edinburgh Univ. Press, 1972, pp. 91–100.

[9] H.-D. Ebbinghaus, J. Flum, and W. Thomas, *Mathematical Logic*, Springer, 2nd ed., 1994.

[10] J. Ferrante and C. Rackoff, *A decision procedure for the first order theory of real addition with order*, SIAM J. Comput., 4 (1975), pp. 69–76.

[11] J. Ferrante and C. W. Rackoff, *The Computational Complexity of Logical Theories*, vol. 718 of Lect. Notes Math., Springer, 1979.

[12] B. Khoussainov and A. Nerode, *Automatic presentations of structures*, in Proc. of the Int. Workshop on Logical and Computational Complexity (LCC), vol. 960 of Lect. Notes Comput. Sci., Springer, 1995, pp. 367–392.

[13] B. Khoussainov, S. Rubin, and F. Stephan, *Definability and regularity in automatic structures*, in Proc. of the 21st Annual Symp. on Theoretical Aspects of Computer Science (STACS), vol. 2996 of Lect. Notes Comput. Sci., Springer, 2004, pp. 440–451.

[14] F. Klaedtke, *On the automata size for Presburger arithmetic*, in Proc. of the 19th Annual IEEE Symp. on Logic in Computer Science (LICS), IEEE Computer Society Press, 2004, pp. 110–119. Accepted for publication in ACM Trans. Comput. Log..

[15] N. Klarlund, A. Møller, and M. I. Schwartzbach, *MONA implementation secrets*, Int. J. Found. Comput. Sci., 13 (2002), pp. 571–586.

[16] D. Kozen, *Theory of Computation*, Springer, 2006.

[17] R. E. Ladner, *Application of model theoretic games to discrete linear orders and finite automata*, Inform. and Control, 33 (1977), pp. 281–303.

[18] C. Löding, *Efficient minimization of deterministic weak ω-automata*, Inform. Process. Lett., 79 (2001), pp. 105–109.

[19] O. Maler and L. Staiger, *On syntactic congruences for omega-languages*, Theoret. Comput. Sci., 181 (1997), pp. 93–112.

[20] A. R. Meyer, *Weak monadic second-order theory of successor is not elementary-recursive*, in Logic Colloquium, vol. 453 of Lect. Notes Math., Springer, 1975, pp. 132–154.

[21] J. Paredaens, J. Van den Bussche, and D. Van Gucht, *First-order queries on finite structures over the reals*, SIAM J. Comput., 27 (1998), pp. 1747–1763.

[22] C. Reddy and D. W. Loveland, *Presburger arithmetic with bounded quantifier alternation*, in Proc. of the 10th Annual ACM Symp. on Theory of Computing (STOC), ACM Press, 1978, pp. 320–325.

[23] L. Stockmeyer, *The complexity of decision problems in automata theory and logic*, PhD thesis, Department of Electrical Engineering, MIT, Boston, MA, USA, 1974.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 457-466
www.stacs-conf.org

NEW COMBINATORIAL COMPLETE ONE-WAY FUNCTIONS

ARIST KOJEVNIKOV [1] AND SERGEY I. NIKOLENKO [1]

[1] St.Petersburg Department of V. A. Steklov Institute of Mathematics
Fontanka 27, St.Petersburg, Russia, 191023

URL: `http://logic.pdmi.ras.ru/{~arist,~sergey}/`

Abstract. In 2003, Leonid A. Levin presented the idea of a combinatorial complete one-way function and a sketch of the proof that Tiling represents such a function. In this paper, we present two new one-way functions based on semi-Thue string rewriting systems and a version of the Post Correspondence Problem and prove their completeness. Besides, we present an alternative proof of Levin's result. We also discuss the properties a combinatorial problem should have in order to hold a complete one-way function.

1. Introduction

In computer science, complete objects play an extremely important role. If a certain class of problems has a complete representative, one can shift the analysis from the whole class (where usually nothing can really be proven) to this certain, well-specified complete problem. Examples include Satisfiability and Graph Coloring for NP (see [GJ79] for a survey) or, which is more closely related to our present work, Post Correspondence and Matrix Transformation problems for DistNP [Gur91, BG95].

However, there are problems that are undoubtedly complete for their complexity classes but do not actually cause such a nice concept shift because they are too hard to analyze. Such problems usually come from diagonalization procedures and require enumeration of all Turing machines or all problems of a certain complexity class.

Our results lie in the field of cryptography. For a long time, little has been known about complete problems in cryptography. While "conventional" complexity classes got their complete representatives relatively soon, it had taken thirty years since the definition of a public-key cryptosystem [DH76] to present a complete problem for the class of all public-key cryptosystems [HKN$^+$05, GHP06]. However, this complete problem is of the "bad" kind of complete problems, requires enumerating all Turing machines and can hardly be put to any use, be it practical implementation or theoretical complexity analysis.

1998 ACM Subject Classification: E.3, F.1.1, F.1.3.
Key words and phrases: cryptography, complete problem, one-way function.
Supported in part by INTAS (YSF fellowship 05-109-5565) and RFBR (grants 05-01-00932, 06-01-00502).

Before tackling public-key cryptosystems, it is natural to ask about a seemingly simpler object: one-way functions (public-key cryptography is equivalent to the existence of a trap-door function, a particular case of a one-way function). The first big step towards useful complete one-way functions was taken by Leonid A. Levin who provided a construction of the first known complete one-way function [Lev87] (see also [Gol99]).

The construction uses a universal Turing machine U to compute the following function:

$$f_{uni}(\text{desc}(M), x) = (\text{desc}(M), M(x)),$$

where $\text{desc}(M)$ is the description of a Turing machine M. If there are one-way functions among M's (and it is easy to show that if there are any, there are one-way functions that run in, say, quadratic time), then f_{uni} is a (weak) one-way function.

As the reader has probably already noticed, this complete one-way function is of the "useless" kind we've been talking about. Naturally, Levin asked whether it is possible to find "combinatorial" complete one-way functions, functions that would not depend on enumerating Turing machines or giving their descriptions as input. For 15 years, the problem remained open and then was resolved by Levin himself [Lev03]. Levin devised a clever trick of having determinism in one direction and indeterminism in the other.

Having showed that a modified Tiling problem is in fact a complete one-way function, Levin asked to find other combinatorial complete one-way functions. In this work, we answer this open question. We take Levin's considerations further to show how a complete one-way function may be derived from string-rewriting problems shown to be average-case complete in [Wan95] and a variation of the Post Correspondence Problem. Moreover, we discuss the general properties a combinatorial problem should enjoy in order to contain a complete one-way function by similar arguments.

2. Distributional Accessibility problem for semi-Thue systems

Consider a finite alphabet $\mathcal{A}$. An ordered pair of strings $\langle g, h\rangle$ over $\mathcal{A}$ is called a *rewriting rule* (sometimes also called a *production*). We write these pairs as $g \to h$ because we interpret them as rewriting rules for other strings. Namely, for two strings u, v we write $u \Rightarrow_{g\to h} v$ if $u = agb$, $v = ahb$ for some $a, b \in \mathcal{A}^*$. A set of rewriting rules is called a *semi-Thue system*. For a semi-Thue system R, we write $u \Rightarrow_R v$ if $u \Rightarrow_{g\to h} v$ for some rewriting rule $\langle g, h\rangle \in R$. Slightly abusing notation, we extend it and write $u \Rightarrow_R v$ if there exists a finite sequence of rewriting rules $\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle \in R$ such that

$$u = u_0 \Rightarrow_{g_1\to h_1} u_1 \Rightarrow_{g_2\to h_2} u_2 \Rightarrow \ldots \Rightarrow_{g_m\to h_m} u_m = v.$$

For a more detailed discussion of semi-Thue systems we refer the reader to [BO93].

We can now define the distributional accessibility problem for semi-Thue systems:

Instance. A semi-Thue system $R = \{\langle g_1, h_1\rangle, \dots, \langle g_m, h_m\rangle\}$, two binary strings u and v, a positive integer n. The size of the instance is $n + |u| + |v| + \sum_1^m (|g_i| + |h_i|)$.
Question. Is $u \Rightarrow_R^n v$?
Distribution. Randomly and independently choose positive integers n and m and binary strings u and v. Then randomly and independently choose binary strings $g_1, h_1, \dots, g_m, h_m$. Integers and strings are chosen with the default uniform probability distribution, namely the distribution proportional to $\frac{1}{n^2}$ for integers and proportional to $\frac{2^{-|u|}}{|u|^2}$ for binary strings.

In [WB95], this problem was shown to be complete for DistNP.

For what follows, we also need another notion of derivation in semi-Thue systems. Namely, for a semi-Thue system R we write $u \Rightarrow_R^* v$ if $u = agb$, $v = ahb$ for some $\langle g, h\rangle \in R$ and, moreover, there does not exist another rewriting rule $\langle g', h'\rangle \in R$ such that $u = a'g'b'$ and $v = a'h'b'$ for some $a', b' \in \mathcal{A}^*$. Similarly to $\Rightarrow_R$, we extend $\Rightarrow_R^*$ to finite chains of derivations. In other words, $u \Rightarrow_R^* v$ if $u \Rightarrow_R v$, and on each step of this derivation there was only one applicable rewriting rule. This uniqueness (or, better to say, determinism) is crucial to perform Levin's trick. We also write $u \Rightarrow_R^{*,n} v$ if $u \Rightarrow_R^* v$ in at most n steps.

3. Post Correspondence Problem

The following problem was proven to be complete for DistNP in [Gur91] (see also Remark 2 in [BG95]):

Instance. A positive integer m, pairs $\Gamma = \{\langle u_1, v_1\rangle, \dots, \langle u_m, v_m\rangle\}$, a binary string x, a positive integer n. The size of the instance is $n + |x| + \sum_1^m (|u_i| + |v_i|)$.
Question. Is $u_{i_1} \cdots u_{i_k} = u v_{i_1} \cdots v_{i_k}$ for some $k \le n$?
Distribution. Randomly and independently choose positive integers n and m and binary string x. Then randomly and independently choose binary strings $u_1, v_1, \dots, u_m, v_m$. Integers and strings are chosen with the default uniform probability distribution.

We need a modification of this problem. Namely, we pose the question as follows: does

$$u_{i_1} \cdots u_{i_k} y = x v_{i_1} \cdots v_{i_k}$$

hold for some y? If we remove the restriction n, this problem is undecidable, but the bounded version is not known to be complete for DistNP.

Given a nonempty list $\Gamma = (\langle u_1, v_1\rangle, \dots, \langle u_m, v_m\rangle)$ of pairs of strings, it will be convenient to view the function based on modified Post Correspondence Problem as a derivation with pairs from Γ as inference rules. A string x *yields* a string y *in one step* if there is a pair $\langle u, v\rangle$ in Γ such that $uy = xv$. The "*yield*" relation $\vdash_\Gamma$ is defined as the transitive closure of the "yield-in-one-step" relation.

To perform Levin's trick, we need to get rid of the indeterminism. This time, the description of a deterministic version of $\vdash^*$ is more complicated than in the case of semi-Thue systems. If we simply required it to be deterministic, we would not be able to move

the head of the Turing machine to the left. To solve this problem, we have to look ahead by one step: if one of the two branches fails in two steps, we consider the choice deterministic.

Formally speaking, we write $x \vdash^* y$ if there are no more than two pairs $\langle p, s\rangle, \langle p', s'\rangle \in \Gamma$ such that $py = xs$ and $p'y' = xs'$ for some strings y, y' (where $y \neq y'$, but p may equal p': two possible different applications of the same rule are still nondeterministic) and, moreover, we cannot apply any rule in Γ to y'. We write $u \vdash_\Gamma^{*,n} v$ if $u \vdash_\Gamma^* v$ in not more than n steps.

4. Complete One-Way Tiling Function

Before presenting our own construction, we recall Levin's complete one-way function from [Lev03]. In fact, we slightly modify Levin's construction and present an alternative proof based on ideas from [Wan99]. The difference with the original Levin's construction is that he considered the tiling function for tiles with marked corners, namely, the corners of tiles, instead of edges, are marked with symbols. In the tiling of an $n \times n$ square, symbols on touching corners of adjacent tiles should match.

A *tile* is a square with a symbol for a finite alphabet $\mathcal{A}$ on each size which may not be turned over or rotated. We assume that there exist infinite copies of each tile. By a *tiling* of an $n \times n$ square we mean a set of n^2 tiles covering the square in which the symbols on the common sides of adjacent tiles are the same.

It will be convenient for us to consider Tiling as a string transformation system. Fix a finite set of tiles T. We say that T *transforms* a string x to y, $|x| = |y|$, if there is a tiling of an $|x| \times |x|$ square with x on the bottom and y on top. We write $x \longrightarrow_T y$ in this case. By a *tiling process* we mean the completion of a partially tiled square by one tile at the time. Similarly to semi-Thue systems, we define $x \longrightarrow_T^* y$ if and only if $x \longrightarrow_T y$ with an additional restriction: we permit the extension of a partially tiled square only if the possible extension is unique.

Definition 4.1. The *Tiling simulating function* (Tiling) is the function $f : \mathcal{A}^* \to \mathcal{A}^*$ defined as follows:

- if the input has the form (T, x) for a finite set of tiles T and a string x, then:
  - if $x \longrightarrow_T^* y$, then $f(T, x) = (T, y)$;
  - otherwise, f returns its input;
- otherwise, f returns its input.

Theorem 4.2. *If one-way functions exist, then Tiling is a weakly one-way function.*

Proof. Let Q be the set of states of a Turing machine M, s be the initial state of M, h — the halting state, π_M — the transition function of M, $\{0, 1, B\}$ — the tape symbols. By \$ we denote the begin marker and by # — the end marker. We also introduce a new symbol for each pair from $Q \times \{0, 1, B\}$. We now present the construction of a tileset T_M.

(1) For each tape symbol $a \in \{0, 1, B\}$ we add

(2) For each $a, b, c \in \{0, 1, B\}$, $q \in Q \setminus \{h\}$, $p \in Q$, if $\pi_M(q, a) = (p, b, R)$ we add

(3) For each $a, b, c \in \{0, 1, B\}$, $q \in Q \setminus \{h\}$, $p \in Q$, if $\pi_M(q, a) = (p, b, L)$ we add

(4) Finally, for \$ and # we add

The following lemma is now obvious.

Lemma 4.3. *For a deterministic Turing machine M that works n^2 steps and its corresponding tiling system T_M,*

$$M(x) = y,\ |x| = |y|,\ \textit{if and only if}\ \$sxB^{n(n-1)}\# \longrightarrow^*_{T_M} \$hyB^{n(n-1)}\#.$$

The rest of the proof closely follows [Gol99]. Suppose that g is a length-preserving one-way function that, on inputs of length n, works for time not exceeding n^2. By Lemma 4.3, there exists a finite system of tiles T_M such that $\$sxB^{n(n-1)}\# \longrightarrow^*_{T_M} \$hyB^{n(n-1)}\#$ is equivalent to $g(x) = y$. Therefore, with constant probability solving Tiling is equivalent to inverting g. ∎

5. A complete one-way function based on semi-Thue systems

Our complete one-way function is based upon the distributional accessibility problem for semi-Thue systems. First, we need to make this decision problem a function and then add Levin's trick in order to assure length-preservation.

Definition 5.1. The *semi-Thue accessibility function* (STAF) is the function $f : \mathcal{A}^* \to \mathcal{A}^*$ that defined as follows:

- if the input has the form $(\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle, x)$, consider the semi-Thue system $\Gamma = (\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle)$ and:
  - if $x \Rightarrow^{*,t}_{\Gamma} y$, $t = |x|^2 + 4|x| + 2$, there are no rewriting rules in Γ that may be applied to y, and $|y| = |x|$, $f(\Gamma, x) = (\Gamma, y)$;
  - otherwise, f returns its input;
- otherwise, f returns its input.

Obviously, STAF is easy to compute: one simply needs to use the first part of the input as a semi-Thue system (if that's impossible, return input) and apply its rules until either there are two rules that apply, or we have worked for $|x|^2 + 4|x| + 2$ steps, or y has been reached and no other rules can be applied. In the first two cases, return input. In the third case, check that $|y| = |x|$ and return (Γ, y) if so and input otherwise.

Theorem 5.2. *If one-way functions exist, then STAF is a weakly one-way function.*

Proof. This time we need to encode Turing machines into the string-rewriting setting. Following [Gur91, WB95, Wan99], we have the following proposition:

Proposition 5.3. *For any finite alphabet $\mathcal{A}$ with $|\mathcal{A}| > 2$ and any pair of binary strings x and y there exists a* dynamic binary coding scheme *of $\mathcal{A}$ with $\{0,1\}$ with the following properties.*

1. *All codes (binary codes of symbols of $\mathcal{A}$) have the same length $l = 2\log|x| + O(1)$.*
2. *Both strings x and y are distinguishable from every code, that is, no code is a substring of x or y.*
3. *If a nonempty suffix z of a code u is a prefix of a code v then $z = u = v$ (one can always distinguish where a code ends and another code begins).*
4. *Strings x and y can be written as a unique concatenation of binary strings* 1, 10, 000, *and* 100 *which are not prefixes of any code.*

Now let us define the semi-Thue system R_M that corresponds to a Turing machine M. The rewriting rules are divided into three parts: $R_M = R_1 \cup R_2 \cup R_3$. Let us denote $\mathcal{B} = \{1, 10, 100, 000\}$ and fix a dynamic binary coding scheme and denote by $\underline{w}$ the encoding of w in this scheme.

R_1 consists of the following rules for each $u \in \mathcal{B}$:

$$\begin{aligned} \underline{s}u &\to \underline{\$us_1}, \\ \underline{s_1}u &\to \underline{us_1}, \\ \underline{us_1\$} &\to \underline{s_2u\$}, \\ \underline{us_2} &\to \underline{s_2u}, \\ \underline{\$s_2} &\to \underline{\$s}. \end{aligned}$$

These rules are needed to rewrite the initial string $\underline{s}x\underline{\$}$ into $\underline{\$sx\$}$. Since x can be uniquely written as $u_1 \dots u_m$ for some $u_i \in \mathcal{B}$, this transformation can be carried out in $2m + 1 \le 2|x| + 1$ steps.

R_2 consists of rewriting rules corresponding to Turing machine instructions. By h we denote the halting state, by s — the initial state, by B — the blank symbol, by Q_M — the set of states of M, by π_M — the transition function of M, and by \$ the begin/end marker. Then R_2 consists of the following pairs:

(1) For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a, b, c \in \{0, 1, B\}$:

$$\pi_M(q, a) = (p, b, R) \quad \Rightarrow \quad qac \to bpc, qa\$ \to bpB\$ \in R_2.$$

(2) For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a, b, d \in \{0, 1, B\}$ and $c \in \{0, 1, \$\}$,

$$\pi_M(q, a) = (p, b, L) \quad \Rightarrow \quad dqac \to pdbc, dqB\$ \to pdbB\$ \in R_2$$

for $a \ne B$, $c \ne \$$, or $b \ne B$.

R_1 and R_2 are completely similar to the construction presented in [Wan99]. The third part of his construction is supposed to reduce the result from $\underline{\$sy\$}$, where y is the result of the Turing machine computation, to the protocol of the Turing machine that is needed to prove that non-deterministic semi-Thue systems are DistNP-hard.

This time we have to deviate from [Wan99]: we need a different set of rules because we actually need the output of the machine, and not the protocol. Thus, our version of R_3

looks like the following:

$$\begin{array}{rcl} \underline{\$hu} & \to & \underline{\$us_5}, \\ \underline{s_5u} & \to & \underline{us_5}, \\ \underline{s_5u\$} & \to & \underline{us_6\$}, \\ \underline{us_6} & \to & \underline{s_6u}, \\ \underline{\$s_6} & \to & \underline{h}. \end{array}$$

This transformation can be carried out in at most $2|y| + 1$ steps.

These rules simply translate $\underline{y}$ back into the original y and add h in front of the output, thus achieving the actual output configuration of the original Turing machine M.

The following lemma is now obvious.

Lemma 5.4. *For a deterministic Turing machine M and its corresponding semi-Thue system R_M,*

$$M(x) = y \text{ if and only if } \underline{s}x\underline{\$} \Rightarrow^{*,t}_{R_M} \underline{hy}\underline{\$},$$

where $t = T + 2|x| + 2|y| + 2$, T being the running time of M on x.

Again, the rest of the proof follows the lines of [Gol99]. There is a constant probability (for the uniform distribution, it is proportional to $\frac{1}{|R|^2 2^{|R|}}$) that any given semi-Thue system appears as the first part of the input. Suppose that g is a length-preserving one-way function. By [Gol99], we can safely assume that there is a Turing machine M_g that computes g and runs in quadratic time. By Lemma 5.4, there exists a semi-Thue system R_M such that $\underline{s}x\underline{\$} \Rightarrow^{*,t}_{R_M} \underline{hy}\underline{\$}$ is equivalent to $g(x) = y$. Therefore, with constant probability solving STAF is equivalent to inverting g. ∎

6. A complete one-way function based on Post Correspondence

In this section, we describe a one-way function based on the Post Correspondence Problem and prove that it is complete. The function is defined as follows.

Definition 6.1. The *Post Transformation function* (PTF) is the function $f : \mathcal{A}^* \to \mathcal{A}^*$ defined as follows:

- if the input has the form $(\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle, x)$, considers the derivation system $\Gamma = (\langle g_1, h_1\rangle, \ldots, \langle g_m, h_m\rangle)$ and:
  - if $x \vdash^{*,n^4}_{\Gamma} y$, there are no rewriting rules in Γ that may be applied to y, and $|y| = |x|$, then $f(\Gamma, x) = (\Gamma, y)$;
  - otherwise, f returns its input;
- otherwise, f returns its input.

Now, we reduce the computation of a universal Turing machine to Post Correspondence in the way described in [Gur91].

Theorem 6.2. *If one-way functions exist, then PTF is a weakly one-way function.*

Proof. As usual, let Q be the set of states of a Turing machine M, s be the initial state of M, h — the halting state, π_M — the transition function of M, $0, 1, B$ — the tape symbols. For all symbols we use the dynamic binary coding scheme described in Section 5.

We now present the construction of a derivation set Γ_M.

(1) For every tape symbol x:
$$\langle \underline{x}, \underline{x} \rangle.$$
(2) For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a, b \in \{0, 1\}$ and rule $\pi_M(q, a) = (p, b, R)$:
$$\langle \underline{qa}, \underline{bp} \rangle.$$
(3) For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a \in \{0, 1\}$ and rule $\pi_M(q, B) = (p, a, R)$:
$$\langle \underline{qB}, \underline{bpB} \rangle.$$
(4) For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a, b, c \in \{0, 1\}$ and rule $\pi_M(q, a) = (p, b, L)$:
$$\langle \underline{cqa}, \underline{pcb} \rangle.$$
(5) For each state $q \in Q_M \setminus \{h\}$, $p \in Q$, $a \in \{0, 1\}$ and rule $\pi_M(q, B) = (p, a, L)$:
$$\langle \underline{cqB}, \underline{pcbB} \rangle.$$

The configuration of M after t steps of computation is represented by a string xqy, where q is the current state of M, x is the tape before the head, and y is the tape from the head to the first blank symbol. The simulation of a step of M from a configuration xqy consists of at most $|x|$ applications of the rule 1, followed by one application of one of the rules 2–5, followed by $|y| - 1$ applications of rule 1. Note that before an application of a rule that moves head to the left one could also apply rule 1. If the Turing Machine M is deterministic, then this "wrong" application leads to a situation where no rule from Γ_M is applicable. Thus, we have the following lemma.

Lemma 6.3. *For a deterministic Turing machine M with running time at most n^2 and its corresponding Post Transformation system Γ_M,*
$$M(x) = y \text{ if and only if } \underline{sxB} \vdash_{\Gamma_M}^{*, n^4} \underline{hyB}.$$

As usual, the rest of the proof closely follows [Gol99]. Suppose that g is a length-preserving one-way function that works for time not exceeding n^2. By Lemma 6.3, there exists a finite system of pair Γ_M such that $\underline{sxB} \vdash_{R_M}^{*, n^4} \underline{hyB}$ is equivalent to $g(x) = y$. Therefore, with constant probability solving PTF is equivalent to inverting g. ∎

Remark 6.4. Note the slight change in distributions on inputs and outputs: PTF accepts as input $\underline{x}$ and outputs $\underline{y}$, while the emulated machine g accepts x and outputs y. Such "tiny details" often hold the devil of average-case reasoning. Fortunately, distributions on x and $\underline{x}$ can be transformed from one to another by a polynomial algorithm, so PTF is still a weak one-way function (see [Gol99] for details).

7. Complete one-way functions and DistNP-hard combinatorial problems

Both our constructions of a complete one-way function look very similar to the construction on the Tiling complete one-way function. This naturally leads to the question: in what other combinatorial settings can one apply the same reasoning to find a complete one-way function?

The whole point of this proof is to keep the function both length-preserving and easily computable. Obvious functions fall into one of two classes.

(1) *Easily computable, but not length-preserving.* For any DistNP-hard problem, one can construct a hard-to-invert function f that transfers *protocols* of this problem into its results. This function is hard to invert on average, but it does not preserve length, and thus it is impossible to translate a uniform distribution on *outputs* of f into a reasonable distribution on its *inputs*. The reader is welcome to think of a reasonably uniform distribution on *proper* tilings that would result in a reasonably uniform distribution on their upper rows; we believe that to construct such a distribution is either impossible or requires a major new insight.
(2) *Length-preserving, but hard to compute.* Take a DistNP-hard problem and consider the function that sends its input into its output (e.g. the lowest row of the tiling into its uppermost row). This function is hard to invert and length-preserving, but it is also hard to compute, because to compute it one needs to solve Tiling.

Following Levin, we get around these obstacles by having a *deterministic* version of a DistNP-hard problem. This time, a Tiling problem produces nontrivial results only if there always is *only one* proper tile to attach. Similarly, in Section 5 we demanded that there is only one rewriting rule that applicable on each step (we introduced $\Rightarrow^*$ for this very purpose). In Section 6 we slightly generalized this idea of determinism, allowing fixed length deterministic backtrack. However, if for all $z \in f^{-1}(y)$ we can do this deterministic procedure, then we can easily invert f. So we need that for most z an indeterminism appears and the procedure return z.

A combinatorial problem should have two properties in order to hold a complete one-way function.

(1) It should have a deterministic restricted version, like Tiling, string rewriting and modified Post Correspondence.
(2) Its deterministic version should be powerful enough to simulate a deterministic Turing machine. For example, natural deterministic Post Correspondence (without any backtrack) is, of course, easy to formulate, but does not seem to be powerful enough.

Keeping in mind these properties, one is welcome to look for other combinatorial settings with combinatorial complete one-way functions.

8. Discussion and further work

We have shown a new complete one-way function and discussed possibilities of other combinatorial settings to hold complete one-way functions. These functions are combinatorial in nature and represent a step towards the easy-to-analyze complete cryptographic objects, much like SAT is a perfect complete problem for NP.

However, we are still not quite there. Basically, we sample a Turing machine at random and hope to find precisely the hard one. This distinction is very important for practical implications of our constructions. We believe that constructing a complete cryptographic problem that has properties completely analogous to SAT requires a major new insight, and such a construction represents one of the most important challenges in modern cryptography.

Another direction would be to find other similar combinatorial problems that can hold a complete one-way function. By looking at our one-way functions and Levin's Tiling, one could imagine that every DistNP-complete problem readily yields a complete one-way function. However, there is also this subtle requirement that the problem (or its appropriate

restriction) should be deterministic (compare $\Rightarrow_R^*$ and $\Rightarrow_R$). It would be interesting to restate this requirement as a formal restriction on the problem setting. This would require some new definitions and, perhaps, a more general and unified approach to combinatorial problems.

Acknowledgments

The authors are very grateful to Dima Grigoriev, Edward A. Hirsch and Yuri Matiyasevich for helpful comments and fruitful discussions.

References

[BG95] Andreas Blass and Yuri Gurevich. Matrix transformation is complete for the average case. *SIAM Journal on Computing*, 24(1):3–29, 1995.

[BO93] Ronald V. Book and Friedrich Otto. *String Rewriting Systems*. Springer-Verlag, 1993.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[GHP06] Dima Grigoriev, Edward A. Hirsch, and Konstantin Pervyshev. A complete public-key cryptosystem. Technical Report 006-046, Electronic Colloquium on Computational Complexity, 2006.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, CA, 1979.

[Gol99] Oded Goldreich. *Introduction to Complexity Theory. Lecture Notes*. Weizmann Institute of Science, 1998-99.

[Gur91] Yuri Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42(3):346–398, 1991.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In *Eurocrypt'05*, pages 96–113, 2005.

[Lev87] Leonid A. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[Lev03] Leonid A. Levin. The tale of one-way functions. *Problems of Information Transmission*, 39(1):92–103, 2003.

[Wan95] Jie Wang. Random instances of bounded string rewriting are hard. *Journal of Computing and Information*, 1(1):11–23, 1995.

[Wan99] Jie Wang. Distributional word problem for groups. *SIAM Journal on Computing*, 28(4):1264–1283, 1999.

[WB95] Jie Wang and Jay Belanger. On the np-isomorphism problem with respect to random instances. *Journal of Computer and System Sciences*, 50(1):151–164, 1995.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 467-478
www.stacs-conf.org

COMPATIBILITY OF SHELAH AND STUPP'S AND MUCHNIK'S ITERATION WITH FRAGMENTS OF MONADIC SECOND ORDER LOGIC

DIETRICH KUSKE

Institut für Informatik, Universität Leipzig

Abstract. We investigate the relation between the theory of the iterations in the sense of Shelah-Stupp and of Muchnik, resp., and the theory of the base structure for several logics. These logics are obtained from the restriction of set quantification in monadic second order logic to certain subsets like, e.g., finite sets, chains, and finite unions of chains. We show that these theories of the Shelah-Stupp iteration can be reduced to corresponding theories of the base structure. This fails for Muchnik's iteration.

1. Introduction

Rabin's tree theorem states, via an automata-theoretic proof, the decidability of the monadic second order (short: MSO) theory of the complete binary tree. It allows to derive the decidability of seemingly very different theories (e.g., the MSO-theory of the real line where set quantification is restricted to closed sets [11]). Its importance is stressed by Seese's result that any class of graphs of bounded degree with a decidable MSO-theory has bounded tree-width (i.e., is "tree-like") [13].

In [15], Shelah reports a generalization of Rabin's tree theorem that was proved by Shelah and Stupp. The idea is to start with a structure $\mathfrak{A}$ and to consider the tree whose nodes are the finite words over the universe of $\mathfrak{A}$ together with the prefix order on these words. Then the immediate successors of any node in this tree can naturally be identified with the elements of the structure $\mathfrak{A}$ – hence they carry the relations of $\mathfrak{A}$. The resulting tree with additional relations is called *Shelah-Stupp-iteration*. The above mentioned result of Shelah and Stupp states that the MSO-theory of the Shelah-Stupp-iteration can be reduced to the MSO-theory of the base structure $\mathfrak{A}$. If $\mathfrak{A}$ is the two-elements set, then Rabin's tree theorem follows.

A further extension is attributed to Muchnik [14] who added a unary clone predicate to Shelah and Stupp's iteration resulting in the *Muchnik-iteration*. This clone predicate states that the last two letters of a word are the same. This allows, e.g., to define the unfolding of a rooted graph in its Muchnik-iteration [5]. Muchnik's theorem then gives a reduction of the MSO-theory of the Muchnik-iteration to the MSO-theory of the base structure. The proof was not published by Muchnik himself, but, using automata-theoretic methods, Walukiewicz

1998 ACM Subject Classification: F.4.1.
Key words and phrases: Logic in computer science, Rabin's tree theorem.

showed that the reduction in Muchnik's theorem is even uniform (i.e., independent from the concrete base structure) [17]. Since, as mentioned above, the unfolding of a rooted graph can be defined in the Muchnik-iteration, the MSO-theory of this unfolding can be reduced to that of the graph [5]. This result forms the basis for Caucal's hierarchy [2] of infinite graphs with a decidable MSO-theory. Walukiewicz's automata-theoretic proof ideas have been shown to work for the Muchnik-iteration and stronger logics like Courcelle's counting MSO and guarded second-order logic by Blumensath & Kreutzer [1].

In [10], we asked for a first-order version of Muchnik's result – and failed. More precisely, we constructed structures with a decidable first-order theory whose Muchnik-iteration has an undecidable first-order theory. As it turns out, the only culprit is Muchnik's clone predicate since, on the positive side, we were able to uniformly reduce the first-order theory (and even the monadic chain theory where set variables range over chains, only) of the Shelah-Stupp-iteration to the first-order theory of the base structure.[1]

The aim of this paper is to clarify the role of weak monadic second order logic $\mathrm{MSO}^{\mathrm{w}}$ in the context of Shelah-Stupp- and Muchnik-iteration. We first define infinitary versions of these iterations that contain, in addition to the finite words, also ω-words. On the positive side, we prove a rather satisfactory relation between the theories of the infinitary Shelah-Stupp-iteration and the base structure. More precisely, the Shelah-Stupp result together with some techniques from [11] allows to uniformly reduce the $\mathrm{MSO}^{\mathrm{closed}}$-theory of the infinitary Shelah-Stupp-iteration (where set quantification is restricted to closed sets) to the MSO-theory of the base set. Our result from [10] ensures that Shelah-Stupp-iteration is FO-compatible in the sense of Courcelle (i.e., the FO-theory of the infinitary Shelah-Stupp-iteration can be reduced uniformly to the FO-theory of the base structure). Our new positive result states that Shelah-Stupp-iteration is also $\mathrm{MSO}^{\mathrm{w}}$-compatible. To obtain this result, one first observes that the finiteness of a set in the Shelah-Stupp-iteration is definable in $\mathrm{MSO}^{\mathrm{mch}}$ (where quantification is restricted to finite unions of chains), hence the $\mathrm{MSO}^{\mathrm{w}}$-theory of the Shelah-Stupp-iteration can be reduced to its $\mathrm{MSO}^{\mathrm{mch}}$-theory. For this logic, we then prove a result analogous to Rabin's basis theorem: Any consistent $\mathrm{MSO}^{\mathrm{mch}}$-property in the Shelah-Stupp-iteration of a finite union of chains (i.e., of a certain set of words over the base structure) has a witness that can be accepted by a small automaton. But an automaton over a fixed set of states can be identified with its transition matrix, i.e., with a fixed number of finite sets in the base structure. We then prove that $\mathrm{MSO}^{\mathrm{mch}}$-properties of the language of an automaton can effectively be translated into $\mathrm{MSO}^{\mathrm{w}}$-properties of the transition matrix.

On the negative side, we prove that infinitary Muchnik-iteration is not $\mathrm{MSO}^{\mathrm{w}}$-compatible. Namely, there is a tree T_ω with decidable $\mathrm{MSO}^{\mathrm{w}}$-theory such that for any set M of natural numbers, there exists an $\mathrm{MSO}^{\mathrm{w}}$-equivalent tree $\mathfrak{A}_M$ such that M can be reduced to the $\mathrm{MSO}^{\mathrm{w}}$-theory of the infinitary Muchnik-iteration of $\mathfrak{A}_M$. This proof uses the fact that the existence of an infinite branch in a tree is not expressible in $\mathrm{MSO}^{\mathrm{w}}$, but it is a first-order (and therefore a $\mathrm{MSO}^{\mathrm{w}}$-) property of the infinitary Muchnik-iteration.

[1] In the meantime, Alexis Bes found a simpler proof of a stronger result based on the ideas of automatic structures and [16] (personal communication).

2. Preliminaries

2.1. Logics

A *(relational) signature* σ consists of finitely many constant and relation symbols (together with the arity of the latter); a *purely relational signature* does not contain any constant symbols. Formulas use *individual* and *set variables*, usually denoted by small and capital, resp., letters from the end of the alphabet. *Atomic formulas* are $x_1 = x_2$, $R(x_1, \dots, x_n)$, and $x_1 \in X$ where R is an n-ary relation symbol from σ, $x_1, x_2, \dots, x_n$ are individual variables or constant symbols, and X is a set variable. *Formulas* are obtained from atomic formulas by conjunction, negation, and quantification $\exists Z$ for Z an individual or a set variable. A *sentence* is a formula without free variables. The satisfaction relation $\models$ between a σ-structure $\mathfrak{A}$ and formulas is defined as usual. For two σ-structures $\mathfrak{A}$ and $\mathfrak{B}$, we write $\mathfrak{A} \equiv_m^{\mathrm{MSO}} \mathfrak{B}$ if, for any sentence φ of quantifier depth at most m, we have $\mathfrak{A} \models \varphi$ iff $\mathfrak{B} \models \varphi$. If $\mathfrak{A}$ and $\mathfrak{B}$ agree on all first-order formulas (i.e., formulas without set quantification) of quantifier depth at most m, then we write $\mathfrak{A} \equiv_m^{\mathrm{FO}} \mathfrak{B}$.

Let $(V, \preceq)$ be a partially ordered set. A set $M \subseteq V$ is a *chain* if $(M, \preceq)$ is linearly ordered, it is a *multichain* if M is a finite union of chains. An element $x \in M$ is a *branching point* if $\{y \in M \mid x < y\}$ is nonempty and does not have a least element.

We will also consider different restrictions of the satisfaction relation $\models$ where set variables range over certain subsets, only. In particular, we will meet the following restrictions.

- Set quantification can be restricted to finite sets, i.e., we will discuss weak monadic second order logic. The resulting satisfaction relation is denoted $\models^{\mathrm{w}}$ and the equivalence of structures $\equiv_m^{\mathrm{w}}$.
- Set quantification can be restricted to chains (where we assume a designated binary relation symbol $\preceq$ in σ) which results in $\models^{\mathrm{ch}}$ and $\equiv_m^{\mathrm{ch}}$, cf. Thomas [16].
- $\models^{\mathrm{mch}}$ etc. refer to the restriction of set quantification to multichains.
- The superscript closed denotes that set variables range over closed sets, only (where we associate a natural topology to any σ-structure), cf. Rabin [11].

Let t be some transformation of σ-structures into τ-structures, e.g., transitive closure. A very strong relation between the $\mathcal{L}$-theory of $\mathfrak{A}$ and the $\mathcal{K}$-theory of $t(\mathfrak{A})$ is the existence of *a single* computable function red that reduces the $\mathcal{K}$-theory of $t(\mathfrak{A})$ to the $\mathcal{L}$-theory of $\mathfrak{A}$ *for any σ-structure* $\mathfrak{A}$. As shorthand for this fact, we say "The transformation t is $(\mathcal{K}, \mathcal{L})$-compatible" or, slightly less precise "The $\mathcal{K}$-theory of $t(\mathfrak{A})$ is *uniformly reducible* to the $\mathcal{L}$-theory of $\mathfrak{A}$." $(\mathcal{K}, \mathcal{K})$-compatible transformations are simply called $\mathcal{K}$-compatible.

Example 2.1. Any MSO-transduction is MSO-compatible [4] and finite set interpretations are $(\mathrm{MSO}^{\mathrm{w}}, \mathrm{FO})$-compatible [3]. Feferman & Vaught showed that any generalized product is FO-compatible [7]. Finally, any generalized sum is MSO-compatible by Shelah [15].

2.2. Shelah and Stupp's and Muchnik's iteration

Let A be a (not necessarily finite) alphabet. With A^* we denote the set of all finite words over A, A^ω is the set of infinite words, and $A^\infty = A^* \cup A^\omega$. The prefix relation on finite and infinite words is $\preceq$. The set of finite prefixes of a word $u \in A^\infty$ is denoted $\downarrow u = \{v \in A^* \mid v \preceq u\}$, if $C \subseteq A^\infty$, then $\downarrow C = \bigcup_{u \in C} \downarrow u$. For $L \subseteq A^\infty$ and $u \in A^*$ let $u^{-1}L = \{v \in A^\infty \mid uv \in L\}$ denote the left-quotient of L with respect to u.

Let σ be a relational signature and let $\mathfrak{A} = (A, (R^{\mathfrak{A}})_{R\in\sigma})$ be a structure over the signature σ. The *infinitary Shelah-Stupp-iteration* $\mathfrak{A}^\infty$ *of* $\mathfrak{A}$ is the structure

$$\mathfrak{A}^\infty = (A^\infty, \preceq, (\widehat{R})_{R\in\sigma}, \varepsilon)$$

where, for $R \in \sigma$,

$$\widehat{R} = \{(ua_1, \ldots, ua_n) \mid u \in A^*, (a_1, \ldots, a_n) \in R^{\mathfrak{A}}\} .$$

The *(finitary) Shelah-Stupp-iteration* $\mathfrak{A}^*$ is the restriction of $\mathfrak{A}^\infty$ to the set of finite words A^*.

Example 2.2. Suppose the structure $\mathfrak{A}$ has two elements a and b and two unary relations $R_1 = \{a\}$ and $R_2 = \{b\}$. Then $\widehat{R}_1 = \{a, b\}^*a$ and $\widehat{R}_2 = \{a, b\}^*b$. Hence the finitary Shelah-Stupp-iteration $\mathfrak{A}^*$ can be visualized as a complete binary tree with unary predicates telling whether the current node is the first or the second son of its father. In addition, the root ε is a constant of the Shelah-Stupp-iteration $\mathfrak{A}^*$. Furthermore, the infinitary Shelah-Stupp-iteration $\mathfrak{A}^\infty$ adds leaves to this tree at the end of any branch. Since this allows to define $(\mathbb{R}, \leq)$ in $\mathfrak{A}^\infty$, the unrestricted MSO-theory of $\mathfrak{A}^\infty$ is undecidable.

Example 2.3. (cf. [9]**)** The Shelah-Stupp iteration allows to reduce the Cayley graph of a free product to the Cayley graphs of the factors. Let $M_i = (M_i, \circ_i, 1_i)$ be monoids finitely generated by Γ_i for $1 \leq i \leq n$ and let $G_i = (M_i, (E_i^a)_{a\in\Gamma_i}, \{1_i\})$ denote the rooted Cayley graph of M_i. Then the Cayley graph $G = (P, (E^a)_{a\in\bigcup\Gamma_i})$ of the free product $P = (P, \circ, 1)$ of these monoids can be defined in the Shelah-Stupp iteration of the disjoint union of the Cayley graphs G_i. For this to work, let $M = \bigcup_{1\leq i\leq n} M_i$ be the disjoint union of the monoids M_i and consider the structure

$$\mathcal{A} = (M, (M_i)_{1\leq i\leq n}, (E_i^a)_{\substack{1\leq i\leq n\\ a\in\Gamma_i}}, U)$$

where $U = \{1_i \mid 1 \leq i \leq n\}$ is the set of units.

Then a word $w \in M^*$ belongs to the direct product P iff the following holds in the Shelah-Stupp iteration of $\mathcal{A}$:

$$\bigwedge_{1\leq i\leq n} \forall x \lessdot y \preceq w : x \in \widehat{M_i} \to y \notin \widehat{M_i} \wedge y \notin \widehat{U}$$

where $\lessdot$ denotes the immediate successor relation of the partial order $\preceq$. For $a \in \Gamma_i$ and $v, w \in P$, we have $v \circ a = w$ (i.e., $(v, w) \in E^a$) iff the Shelah-Stupp iteration satisfies

$$\left(\exists v' \in \widehat{U} : v \lessdot v' \wedge (v', w) \in \widehat{E_i^a}\right) \vee (v, w) \in \widehat{E_i^a} \vee \left(\exists w' \in \widehat{U} : w \lessdot w' \wedge (v, w') \in \widehat{E_i^a}\right) .$$

Muchnik introduced the additional unary *clone predicate* $\mathrm{cl} = \{uaa \mid u \in A^*, a \in A\}$. The extension of the Shelah-Stupp-iterations by this clone predicate will be called *finitary and infinitary Muchnik-iteration* $(\mathfrak{A}^*, \mathrm{cl})$ and $(\mathfrak{A}^\infty, \mathrm{cl})$, resp. Courcelle and Walukiewicz [5] showed that the unfolding of a directed rooted graph G can be defined in the Muchnik iteration (G^*, cl) of G.

To simplify notation, we will occasionally omit the word "finitary" and just speak of the Shelah-Stupp- and Muchnik-iteration.

3. A basis theorem for $\mathrm{MSO}^{\mathrm{mch}}$

Rabin's tree theorem [11] states the decidability of the monadic second order theory of the complete binary tree. As a *corollary* of his proof technique by tree automata, one obtains Rabin's basis theorem [12, Theorem 26]: Let φ be a formula with free variables $X_1, \ldots, X_\ell$ and let $L_1, \ldots, L_\ell \subseteq \{a,b\}^*$ be regular languages such that the binary tree satisfies $\varphi(L_1, \ldots, L_\ell)$. Then it satisfies $\psi(L_1, \ldots, L_\ell)$ where ψ is obtained from φ by restricting all quantifications to regular sets. To obtain this basis theorem, it suffices to show that validity of $\exists X_\ell : \varphi(L_1, \ldots, L_{\ell-1}, X_\ell)$ implies the existence of a regular set R_ℓ such that $\varphi(L_1, \ldots, L_{\ell-1}, R_\ell)$ holds true in the binary tree.

This is precisely what this section shows in our context of $\mathrm{MSO}^{\mathrm{mch}}$ and the Shelah-Stupp-iteration $\mathfrak{A}^*$. Even more, we will not only show that the set R_ℓ can be chosen regular, but we will also bound the size of the automaton accepting it.

Throughout this section, σ denotes some purely relational signature.

3.1. Preliminaries

For $k, \ell \in \mathbb{N}$, let $\tau_{k,\ell}$ be the extension of the signature $(\sigma, \preceq)$ by k constants and ℓ unary relations. Using Hintikka-formulas (see [6] for the definition and properties of these formulas) one can show that for any of the signatures $\tau_{k,\ell}$ and $m \in \mathbb{N}$, there are only finitely many equivalence classes of $\equiv_m^{\mathrm{mch}}$. An upper bound $T(\ell, m)$ for the number of equivalence classes of $\equiv_m^{\mathrm{mch}}$ on formulas over the signature $\tau_{2,\ell}$ can be computed effectively.

Now let $\mathfrak{A} = (A, (R)_{R\in\sigma})$ be some σ-structure. For $u \in A^*$, let $\mathfrak{A}_u^*$ denote the $\tau_{1,0}$-structure $(uA^*, \sqsubseteq, (\overline{R})_{R\in\sigma}, u)$ where

- the relation $\sqsubseteq$ is the restriction of $\preceq$ to uA^* and
- $\overline{R}$ is the restriction of $\widehat{R}$ to uA^+.

For any $u, v \in A^*$, the mapping $f : \mathfrak{A}_u^* \to \mathfrak{A}_v^*$ with $f(ux) = vx$ is an isomorphism – this is the reason to consider $\overline{R}$ and not the restriction of $\widehat{R}$ to uA^*. Similarly, the $\tau_{2,0}$-structure $\mathfrak{A}_{u,v}^* = (uA^* \setminus vA^+, \sqsubseteq, (\overline{R})_{R\in\sigma}, u, v)$ is defined for $u, v \in A^*$ with $u \preceq v$. Here, again, $\overline{R}$ is the restriction of $\widehat{R}$ to $uA^+ \setminus vA^+$.

Frequently, we will consider the structure $\mathfrak{A}^*$ together with some additional unary predicates $L_1, \ldots, L_\ell$. As for the plain structure $\mathfrak{A}^*$, we will also meet the restriction of $(\mathfrak{A}^*, L_1, \ldots, L_\ell)$ to the set uA^*, i.e., the structure $(\mathfrak{A}_u^*, L_1 \cap uA^*, \ldots, L_\ell \cap uA^*)$. To simplify notation, this will be denoted $(\mathfrak{A}_u^*, L_1, \ldots, L_\ell)$; the structure $(\mathfrak{A}_{u,v}^*, L_1, \ldots, L_\ell)$ is to be understood similarly.

Example 2.2 (continued). In the case of Example 2.2, $\mathfrak{A}_u^*$ is just the subtree rooted at the node u. On the other hand, $\mathfrak{A}_{u,v}^*$ is obtained from $\mathfrak{A}_u^*$ by deleting all descendants of v and marking the node v as a constant. Thus, we can think of $\mathfrak{A}_{u,v}^*$ as a tree with a marked leaf. These *special trees* are fundamental in the work of Gurevich & Shelah [8] and of Thomas [16].

In the following, fix some $\ell \in \mathbb{N}$. We then define the operations of product and infinite product of $\tau_{k,\ell}$-structures: If $\mathfrak{A} = (A, \preceq^{\mathfrak{A}}, (R^{\mathfrak{A}})_{R\in\sigma}, a_1, a_2, L_1^{\mathfrak{A}}, \ldots, L_\ell^{\mathfrak{A}})$ is a $\tau_{2,\ell}$-structure and $\mathfrak{B} = (B, \preceq^{\mathfrak{B}}, (R^{\mathfrak{B}})_{R\in\sigma}, b_1, \ldots, b_k, L_1^{\mathfrak{B}}, \ldots, L_\ell^{\mathfrak{B}})$ a disjoint $\tau_{k,\ell}$-structure with $k \geq 1$, then their *product* $\mathfrak{A} \cdot \mathfrak{B}$ is a $\tau_{k,\ell}$-structure. It is obtained from the structure

$$(A \cup B, \preceq^{\mathfrak{A}} \cup \preceq^{\mathfrak{B}}, (R^{\mathfrak{A}} \cup R^{\mathfrak{B}})_{R\in\sigma}, L_1^{\mathfrak{A}} \cup L_2^{\mathfrak{B}}, \ldots, L_\ell^{\mathfrak{A}} \cup L_\ell^{\mathfrak{B}})$$

by identifying a_2 and b_1, taking the transitive closure of the partial orders, and extending the resulting structure by the list of constants $a_1, b_2, b_3, \ldots, b_k$. Now let $\mathfrak{A}_n$ be disjoint $\tau_{2,\ell}$-structures with constants u_n and v_n for $n \in \mathbb{N}$. Then the *infinite product* $\prod_{n\in\mathbb{N}} \mathfrak{A}_n$ is a $\tau_{1,\ell}$-structure. It is obtained from the disjoint union of the structures $\mathfrak{A}_n$ by identifying v_n and u_{n+1} for any $n \in \mathbb{N}$. The only constant of this infinite product is u_0. If $\mathfrak{A} \cong \mathfrak{A}_n$ for all $n \in \mathbb{N}$, then we write simply $\mathfrak{A}^\omega$ for the infinite product of the structures $\mathfrak{A}_n$.

Standard applications of Ehrenfeucht-Fraïssé-games (see [6]) yield:

Proposition 3.1. *Let $j, \ell, m \in \mathbb{N}$, $\mathfrak{A}_n, \mathfrak{A}'_n$ be $\tau_{2,\ell}$-structures for $n \in \mathbb{N}$ and let $\mathfrak{B}, \mathfrak{B}'$ be some $\tau_{j+1,\ell}$-structures such that $\mathfrak{A}_n \equiv_m^{\mathrm{mch}} \mathfrak{A}'_n$ for $n \in \mathbb{N}$ and $\mathfrak{B} \equiv_m^{\mathrm{mch}} \mathfrak{B}'$. Then*

$$\mathfrak{A}_0 \cdot \mathfrak{B} \equiv_m^{\mathrm{mch}} \mathfrak{A}'_0 \cdot \mathfrak{B}' \quad \text{and} \quad \prod_{n\in\mathbb{N}} \mathfrak{A}_n \equiv_m^{\mathrm{mch}} \prod_{n\in\mathbb{N}} \mathfrak{A}'_n \ .$$

Remark 3.2. We sketch a typical use of the above proposition in this section. Let $x \in A^*$ be some sufficiently long word. Since $\equiv_m^{\mathrm{mch}}$ has only finitely many equivalence classes, there exist words u, v, w with $x = uvw$ and $v \neq \varepsilon$ such that $(\mathfrak{A}^*_u, \{x\}) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*_{uv}, \{x\})$. Hence we obtain

$$(\mathfrak{A}^*, \{x\}) = (\mathfrak{A}^*_{\varepsilon,u}, \emptyset) \cdot (\mathfrak{A}^*_u, \{uvw\}) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*_{\varepsilon,u}, \emptyset) \cdot (\mathfrak{A}^*_{uv}, \{uvw\}) \cong (\mathfrak{A}^*, \{uw\}) \ .$$

(This proves that every consistent property of a single element of $\mathfrak{A}^*$ is witnessed by some "short" word.)

The last isomorphism does not hold for the Muchnik-iteration since the clone predicate allows to express that the last letter of u and the first letter of v are connected by some edge in the graph $\mathfrak{A}$.

Convention 3.3. We consider complete deterministic finite automata $\mathcal{M} = (Q, B, \iota, \delta, F)$, called *automata* for short. Its language is denoted $L(\mathcal{M})$. We will also write $p.w$ for $\delta(p, w)$. The *transition matrix* of $\mathcal{M}$ is the tuple $T = (T_{p,q})_{p,q\in Q}$ with $T_{p,q} = \{b \in B \mid \delta(p, b) = q\}$.

As explained above, we will use automata to describe subsets of the Shelah-Stupp iteration $\mathfrak{A}^*$, i.e., the alphabet B will always be a finite subset of the universe of $\mathfrak{A}$. These regular subsets have the following nice property whose proof is obvious.

Lemma 3.4. *Let $\mathfrak{A}$ be a σ-structure with universe A and let $\mathcal{M} = (Q, B, \iota, \delta, F)$ be an automaton with alphabet $B \subseteq A$. Then, for any $u, v \in B^*$ with $\delta(\iota, u) = \delta(\iota, v)$, the mapping $f_{u,v} : uA^* \to vA^* : ux \mapsto vx$ is an isomorphism from $(\mathfrak{A}^*_u, L(\mathcal{M}))$ onto $(\mathfrak{A}^*_v, L(\mathcal{M}))$.*

As a consequence, the number of isomorphism classes of structures $(\mathfrak{A}^*_v, L(\mathcal{M}))$ is finite. This fails in the Muchnik-iteration even for $L(\mathcal{M}) = \emptyset$: With $\mathfrak{A} = (\mathbb{N}, \mathrm{succ})$ and $m, n \in \mathbb{N}$, we have $(\mathfrak{A}^*_m, \mathrm{cl}) \cong (\mathfrak{A}^*_n, \mathrm{cl})$ iff $m = n$ since the structure $(\mathbb{N}, \mathrm{succ}, m)$ can be defined in $(\mathfrak{A}^*_m, \mathrm{cl})$.

3.2. Quantification

While multichains in the Shelah-Stupp-iteration can be rather complicated, this section shows that, up to logical equivalence, we can restrict attention to "simple" multichains. Here, "simple" means that they are regular and, even more, can be accepted by a "small" automaton.

For the rest of this section, let $\mathfrak{A} = (A, (R)_{R\in\sigma})$ be some fixed σ-structure and $\ell, m \in \mathbb{N}$. For $1 \leq i \leq \ell$, let $\mathcal{M}_i = (Q_i, B_i, \iota_i, F_i)$ be automata with $B_i \subseteq A$ such that $L(\mathcal{M}_i) \subseteq A^*$

is a multichain in the Shelah-Stupp iteration $\mathfrak{A}^*$. *Write* $\overline{L}$ *for the tuple of multichains* $(L(\mathcal{M}_1), \ldots, L(\mathcal{M}_\ell))$.

Proposition 3.5. *Let* $C \subseteq A^*$ *be a chain. Then there exist* $u, v \in A^*$, $E \subseteq {\downarrow}u \setminus \{u\}$, *and* $F \subseteq {\downarrow}v \setminus \{v\}$ *such that* $\iota_i.u = \iota_i.uv$ *for all* $1 \leq i \leq \ell$ *and* $(\mathfrak{A}^*, \overline{L}, C) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*, \overline{L}, D)$ *with* $D = E \cup uv^*F$.

Proof. One shows the existence of $u_1 \prec u_2 \in A^*$ such that $C \cup \{u_1, u_2\}$ is a chain, $\iota_i.u_1 = \iota_i.u_2$ for all $1 \leq i \leq \ell$, $(\mathfrak{A}^*, \overline{L}) \cong (\mathfrak{A}^*_{\varepsilon,u_1}, \overline{L}) \cdot (\mathfrak{A}^*_{u_1,u_2}, \overline{L})^\omega$, and $(\mathfrak{A}^*, \overline{L}, C) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*_{\varepsilon,u_1}, \overline{L}, C) \cdot (\mathfrak{A}^*_{u_1,u_2}, \overline{L}, C)^\omega$. This uses arguments similar to those in Remark 3.2 and Ramsey's theorem. The result follows with $u = u_1$, $uv = u_2$, $E = C \cap {\downarrow}u \setminus \{u\}$, and $F = u^{-1}(C \cap {\downarrow}u_2 \setminus \{u_2\})$. ∎

The above proposition shows that every consistent property of a chain is witnessed by some regular chain D. Using the pigeonhole principle and arguments as in Remark 3.2, one can bound the lengths of u and v to obtain

Proposition 3.6. *Let* $C \subseteq A^*$ *be a chain. Then there exists an automaton* $\mathcal{N}$ *with at most* $2\prod_{1\leq i\leq \ell} |Q_i| \cdot T(\ell+1, m)$ *states such that* $L(\mathcal{N})$ *is a chain and* $(\mathfrak{A}^*, \overline{L}, C) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*, \overline{L}, L(\mathcal{N}))$.

It is our aim to prove a similar result for arbitrary multichains in place of the chain C in the proposition above. Certainly, in order to get a small automaton for a multichain, the branching points of this set have to be short words. Again using arguments as in Remark 3.2, one obtains

Lemma 3.7. *Let* $M \subseteq A^*$ *be a multichain. Then there exists a multichain* $N \subseteq A^*$ *such that*

- $(\mathfrak{A}^*, \overline{L}, M) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*, \overline{L}, N)$ *and*
- *any branching point of* N *has length at most* $k = \prod_{1\leq i\leq \ell}(|Q_i| + 1) \cdot T(\ell + 1, m)$.

Lemma 3.8. *Let* M *be a multichain such that all branching points of* M *have length at most* $s-1$. *Then there exists an automaton* $\mathcal{N}$ *with at most* $(2\prod_{1\leq i\leq \ell} |Q_i| \cdot T(\ell+1, m))^{s+1}$ *many states such that* $L(\mathcal{N})$ *is a multichain and* $(\mathfrak{A}^*, \overline{L}, M) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*, \overline{L}, L(\mathcal{N}))$.

Proof. Let $n = \prod_{1\leq i\leq \ell} |Q_i|$ and $\overline{L} = (L(\mathcal{M}_1), \ldots, L(\mathcal{M}_\ell))$.

The lemma is shown by induction on s. If $s = 0$, then M is a chain, i.e., the result follows from Prop. 3.6.

Now let M be a multichain such that any branching point has length at most $s > 0$. By the induction hypothesis, for every $a \in A$, there exists an automaton $\mathcal{N}_a$ with at most $(2nT(\ell+1, m))^{s+1}$ many states such that $L(\mathcal{N}_a)$ is a multichain and

$$(\mathfrak{A}^*_a, \overline{L}, M) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*, a^{-1}L(\mathcal{M}_1), \ldots, a^{-1}L(\mathcal{M}^a_\ell), L(\mathcal{N}_a)) \ .$$

Let θ be the equivalence relation on A with $(a, b) \in \theta$ if and only if

(1) $\delta_i(\iota_i, a) = \delta_i(\iota_i, b)$ for all $1 \leq i \leq \ell$ and
(2) $(\mathfrak{A}^*_a, \overline{L}, M) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*_b, \overline{L}, M)$.

Let $H \subseteq A$ contain precisely one element h from any θ-equivalence class. Then the set $\bigcup\{aL(\mathcal{N}_h) \mid a\,\theta\,h \in H \text{ and } a^{-1}M \neq \emptyset\} \cup (\{\varepsilon\} \cap M)$ is a multichain and can be accepted by some automaton $\mathcal{N}$ with the right number of states.

Then $(\mathfrak{A}^*, \overline{L}, L(\mathcal{N}))$ is obtained from $(\mathfrak{A}^*, \overline{L}, M)$ by replacing any subtree $(\mathfrak{A}^*_a, \overline{L}, M)$ with the equivalent structure $(\mathfrak{A}^*, a^{-1}L(\mathcal{M}_1), \ldots, a^{-1}L(\mathcal{M}^a_\ell), L(\mathcal{N}_h))$ for $a\,\theta h \in H$. Hence, by Prop. 3.1, $(\mathfrak{A}^*, \overline{L}, M) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*, \overline{L}, L(\mathcal{N}))$. ∎

Putting these two lemmas together, we obtain that, indeed, every consistent property of a multichain M is witnessed by some multichain that can be accepted by some "small" automaton:

Proposition 3.9. *Let $M \subseteq A^*$ be some multichain. Then there exists an automaton $\mathcal{N}$ with at most $(2nT(\ell+1,m))^{s+1}$ many states (where $s = n \cdot T(\ell+1,m)$, $n = \prod_{1\leq i\leq \ell} |Q_i|$) such that $L(\mathcal{N})$ is a multichain and $(\mathfrak{A}^*, \overline{L}, M) \equiv_m^{\mathrm{mch}} (\mathfrak{A}^*, \overline{L}, L(\mathcal{N}))$.*

Now a result analogous to Rabin's basis theorem follows immediately

Theorem 3.10. *Let $\mathfrak{A}$ be a σ-structure, let φ be an $\mathrm{MSO}^{\mathrm{mch}}$-formula in the language of the Shelah-Stupp-iteration $\mathfrak{A}^*$ with free variables $X_1, \dots, X_\ell$ and let $L_1, \dots, L_\ell \subseteq A^*$ be regular languages such that $(\mathfrak{A}^*, L_1, \dots, L_\ell) \models^{\mathrm{mch}} \varphi$. Then $(\mathfrak{A}^*, L_1, \dots, L_\ell) \models^{\mathrm{reg-mch}} \varphi$ where $\models^{\mathrm{reg-mch}}$ denotes that set quantification is restricted to regular multichains.*

Recall that Rabin's basis theorem follows from his tree theorem whose proof, in turn, uses the effective complementation of Rabin tree automata. While the above theorem is an analogue of Rabin's basis theorem, the proof is more direct and does in particular not rest on any complementation of automata.

4. Shelah-Stupp-iteration is $(\mathrm{MSO}^{\mathrm{mch}}, \mathrm{MSO}^{\mathrm{w}})$-compatible

The results of the previous section, as explained at the beginning, imply that quantification in an $\mathrm{MSO}^{\mathrm{mch}}$-sentence can be restricted to regular sets that are accepted by "small" automata. In this section, we will use this insight to reduce the $\mathrm{MSO}^{\mathrm{mch}}$-theory of the Shelah-Stupp-iteration to the $\mathrm{MSO}^{\mathrm{w}}$-theory of the base structure.

Fix some σ-structure $\mathfrak{A}$ with universe A, some finite set of states Q, some initial state ι, and some set of final states $F \subseteq Q$. Then, for any automaton $\mathcal{M} = (Q, B, \iota, \delta, F)$ with $B \subseteq A$, the language $L(\mathcal{M})$ is a set in the Shelah-Stupp-iteration $\mathfrak{A}^*$ while its transition matrix is a tuple of finite sets in the base structure $\mathfrak{A}$. *The idea of our reduction is that $\mathrm{MSO}^{\mathrm{mch}}$-properties of the set $L(\mathcal{M})$ in the Shelah-Stupp-iteration $\mathfrak{A}^*$ can (effectively) be translated into $\mathrm{MSO}^{\mathrm{w}}$-properties of the transition matrix T in the base structure $\mathfrak{A}$.*

In precisely this spirit, the following lemma expresses simple properties of the automaton $\mathcal{M}$ and of the language $L(\mathcal{M})$ in terms of FO-properties of $(\mathfrak{A}, T) = (\mathfrak{A}, (T_{p,q})_{p,q\in Q})$.

Lemma 4.1. *Let $F \subseteq Q$ be finite sets and $\iota \in Q$. There exist formulas $\mathrm{reach}_{(Q,p,q)}$ for $p, q \in Q$ and $\mathrm{mchain}_{(Q,\iota,F)}$ of FO with free variables $T_{p,q}$ for $p, q \in Q$ such that for any σ-structure $\mathfrak{A}$ and any automaton $\mathcal{M} = (Q, B, \iota, \delta, F)$ with transition matrix T:*
(1) $(\mathfrak{A}, T) \models^{\mathrm{w}} \mathrm{reach}_{(Q,p,q)}$ iff there exists a word $w \in A^*$ with $\delta(p, w) = q$.
(2) $(\mathfrak{A}, T) \models^{\mathrm{w}} \mathrm{mchain}_{(Q,\iota,F)}$ iff $L(\mathcal{M})$ is a multichain.

Proof. The proof is based on the observation that (1) one only needs to search for a path of length at most $|Q|$ and (2) that $L(\mathcal{M})$ is a multichain iff no branching point belongs to some cycle. ∎

So far, we showed that simple properties of $L(\mathcal{M})$ are actually FO- (and therefore $\mathrm{MSO}^{\mathrm{w}}$-) properties of the transition matrix of $\mathcal{M}$. We now push this idea further and consider arbitrary $\mathrm{MSO}^{\mathrm{mch}}$-properties of a tuple of languages $L(\mathcal{M}_1), \dots, L(\mathcal{M}_\ell)$.

Theorem 4.2. *There is an algorithm with the following specification*

input:
- $\ell \in \mathbb{N}$,
- *finite sets* $F_i \subseteq Q_i$ *and states* $\iota_i \in Q_i$ *for* $1 \leq i \leq \ell$,
- *and a formula* α *with free variables among* $L_1, \ldots, L_\ell$ *in the language of the Shelah-Stupp-iteration* $\mathfrak{A}^*$.

output: *A formula* $\alpha_{(\overline{Q},\overline{\iota},\overline{F})}$ *in the language of* $\mathfrak{A}$ *with free variables among* $T^i_{p,q}$ *for* $p, q \in Q_i$ *and* $1 \leq i \leq \ell$ *with the following property:*
If $\mathfrak{A}$ *is a* σ*-structure and* $\mathcal{M}_i = (Q_i, B_i, \iota_i, T^i, F_i)$ *are automata with* $B_i \subseteq A$ *for* $1 \leq i \leq \ell$, *then*

$$(\mathfrak{A}^*, L(\mathcal{M}_1), L(\mathcal{M}_2), \ldots, L(\mathcal{M}_\ell)) \models^{\mathrm{mch}} \alpha \iff (\mathfrak{A}, T^1, T^2, \ldots, T^\ell) \models^{\mathrm{w}} \alpha_{(\overline{Q},\overline{\iota},\overline{F})} .$$

Proof. The proof proceeds by induction on the construction of the formula α, we only sketch the most interesting part $\alpha = \exists X\, \beta$. Set $n = \prod_{1 \leq i \leq \ell} |Q_i|$, $s = nT(\ell + 1, m)$, and $k = (2nT(\ell+1, m))^{s+1}$. Let $\mathfrak{A}$ be a σ-structure and let $\mathcal{M}_i = (Q_i, B_i, \iota_i, \delta_i, F_i)$ be automata with $B_i \subseteq A$ and transition matrix T^i. Then, by Prop. 3.9, $(\mathfrak{A}^*, L(\mathcal{M}_1), \ldots, L(\mathcal{M}_\ell)) \models^{\mathrm{mch}} \alpha$ iff there exists an automaton $\mathcal{N}$ with k states such that

$$(\mathfrak{A}^*, L(\mathcal{M}_1), L(\mathcal{M}_2), \ldots, L(\mathcal{M}_\ell), L(\mathcal{N})) \models^{\mathrm{mch}} \beta .$$

Using the induction hypothesis on β and $\beta_{(\overline{Q},\overline{\iota},\overline{F})}$, this is the case if and only if there exist finite sets $T^{\ell+1}_{i,j}, B \subseteq A$ for $i, j \in [k] = \{1, 2, \ldots, k\}$ such that

- $T^{\ell+1}$ forms the transition matrix of some automaton with alphabet B
- for some $F \subseteq [k]$, the automaton $\mathcal{M}_{\ell+1} = ([k], B, 1, T^{\ell+1}, F)$
  - accepts a multichain M (i.e., $(\mathfrak{A}, T^{\ell+1}) \models^{\mathrm{w}} \mathrm{mchain}_{([k],1,F)}$) and
  - this multichain satisfies β (i.e., $\mathfrak{A}, T^1, \ldots, T^{\ell+1} \models^{\mathrm{w}} \beta_{((\overline{Q},[k]),(\overline{\iota},1),(\overline{F},F))}$).

Since all these properties can be expressed in $\mathrm{MSO}^{\mathrm{w}}$, the construction of $\alpha_{(\overline{Q},\overline{\iota},\overline{F})}$ is complete. ∎

As an immediate consequence, we get a uniform version of Shelah and Stupp's theorem for the logics $\mathrm{MSO}^{\mathrm{w}}$ and $\mathrm{MSO}^{\mathrm{mch}}$:

Theorem 4.3. *Finitary Shelah-Stupp-iteration is* $(\mathrm{MSO}^{\mathrm{mch}}, \mathrm{MSO}^{\mathrm{w}})$*-compatible.*

Remark 4.4. $(\mathrm{MSO}^{\mathrm{ch}}, \mathrm{FO})$-compatibility of Shelah-Stupp-iteration [10] can alternatively be shown along the same lines: One allows incomplete automata and proves an analogue of Prop. 3.6 for the logic $\mathrm{MSO}^{\mathrm{ch}}$. Then Theorem 4.3 can be shown for the pair of logics $(\mathrm{MSO}^{\mathrm{ch}}, \mathrm{FO})$.

5. Infinitary Muchnik-iteration is not $(\mathrm{FO}, \mathrm{MSO}^{\mathrm{w}})$-compatible

Our argument goes as follows: From a set $M \subseteq \mathbb{N}$, we construct a tree $\mathfrak{A}_M$. The $\mathrm{MSO}^{\mathrm{w}}$-theory of this tree will be independent from M and M will be FO-definable in the infinitary Muchnik-iteration $(\mathfrak{A}^\infty_M, \mathrm{cl})$. Assuming $(\mathrm{FO}, \mathrm{MSO}^{\mathrm{w}})$-compatibility of the infinitary Muchnik-iteration, the set M will be reduced uniformly to the $\mathrm{MSO}^{\mathrm{w}}$-theory of $\mathfrak{A}_M$. For $M \neq N$, this yields a contradiction.

A *tree* is a structure $(V, \preceq, r)$ where $\preceq$ is a partial order on V such that, for any $v \in V$, $(\downarrow v, \preceq)$ is a finite linear order and $r \preceq v$ for all $v \in V$.

We will consider the set $T_\omega = \{(a_1, m_1)(a_2, m_2)\dots(a_k, m_k) \in (\mathbb{N}\times\mathbb{N})^* \mid m_1 > m_2 > m_3 \cdots > m_k\}$ of sequences in $\mathbb{N}^2$ whose second components decrease. This set, together with the prefix relation $\preceq$, forms a tree $(T_\omega, \preceq, \varepsilon)$ with root ε that we also denote T_ω. Nodes of the form $w(a,0)$ are leaves of T_ω. Any inner node of T_ω has infinitely many children (among them, there are infinitely many leaves). Furthermore, all the branches of T_ω are finite. Even more, if x is a node different from the root, then the branches passing through x have bounded length.

We will also consider the set $T_\infty = a^*T_\omega$ where a is an arbitrary symbol. Together with the prefix relation, this yields another tree $(T_\infty, \preceq, \varepsilon)$ that we denote T_∞. Differently from T_ω, it has an infinite branch, namely the set of all nodes a^n for $n \in \mathbb{N}$.

For two trees S and T and a node v of S, let $S \cdot_v T$ denote the tree obtained from the disjoint union of S and T by identifying v with the root of T (i.e., the node v gets additional children, namely the children of the root in T).

It is important for our later arguments that this operation transforms trees equivalent wrt. $\equiv^{\mathrm{w}}_m$ into equivalent structures. More precisely

Proposition 5.1. *Let S, T, and T' be trees and $k \in \mathbb{N}$ such that $T \equiv^{\mathrm{w}}_k T'$. Then $S \cdot_v T \equiv^{\mathrm{w}}_k S \cdot_v T'$ for any node v of S.*

With $a^{\leq n} = \{\varepsilon, a, a^2, \dots, a^n\}$, the set $a^{\leq n}T_\omega$ together with the prefix relation and the root, is considered as a tree that we denote $a^{\leq n}T_\omega$.

Proposition 5.2. *For any $k \in \mathbb{N}$, we have $T_\omega \equiv^{\mathrm{w}}_k T_\infty$.*

Proof. The statement is shown by induction on k where the base case $k = 0$ is trivial. To show $T_\omega \equiv^{\mathrm{w}}_{k+1} T_\infty$, it suffices to prove for any formula $\varphi(X)$ of quantifier-depth at most k

$$T_\omega \models^{\mathrm{w}} \exists X\, \varphi(X) \iff T_\infty \models^{\mathrm{w}} \exists X\, \varphi(X)\ .$$

Assuming $T_\infty \models^{\mathrm{w}} \exists X\, \varphi$, there exist $n \in \mathbb{N}$ and $M \subseteq a^{\leq n}T_\omega$ finite with $(T_\infty, M) \models^{\mathrm{w}} \varphi$. Hence we have

$$\begin{aligned}(T_\infty, M) &\cong (a^{\leq n}T_\omega, M) \cdot_{a^n} (T_\infty, \emptyset)\\ &\equiv^{\mathrm{w}}_k (a^{\leq n}T_\omega, M) \cdot_{a^n} (T_\omega, \emptyset) \text{ by Prop. 5.1 and the induction hypothesis}\\ &\cong (a^{\leq n}T_\omega, M)\ .\end{aligned}$$

Hence $(a^{\leq n}T_\omega, M) \models^{\mathrm{w}} \varphi$ and therefore $a^{\leq n}T_\omega \models^{\mathrm{w}} \exists X\, \varphi$. Using $T_\omega \equiv^{\mathrm{w}}_{k+1} a^{\leq n}T_\omega$ (see complete paper for the proof), we obtain $T_\omega \models^{\mathrm{w}} \exists X\, \varphi$.

Conversely, one can argue similarly again using $T_\omega \equiv^{\mathrm{w}}_{k+1} a^{\leq n}T_\omega$. ∎

Remark 5.3. This proves that the existence of an infinite path cannot be expressed in weak monadic second order logic since T_∞ has such a path and T_ω does not.

Using an idea from [5], the existence of an infinite path is a first-order property of the infinitary Muchnik-iteration. The following lemma pushes this idea a bit further:

Lemma 5.4. *Let $T = (T, \leq, r)$ be a tree and let $U \subseteq T$ be the union of all infinite branches of T. Then the* $\mathrm{MSO}^{\mathrm{w}}$*-theory of $(T, \leq, r, U)$ is uniformly reducible to the* $\mathrm{MSO}^{\mathrm{w}}$*-theory of the infinitary Muchnik-iteration (T^∞, cl) of the tree $(T, \leq, r)$ without the extra predicate.*

For $M \subseteq \mathbb{N}$, let $A_M = \{b^m \mid m \in M\}T_\infty \cup \{b^m \mid m \notin M\}T_\omega$ and $\mathfrak{A}_M = (A_M, \preceq, \varepsilon)$. Then $\mathfrak{A}_M$ is obtained from the linear order $(\mathbb{N}, \leq) \cong (b^*, \preceq)$ by attaching the tree T_∞ to elements from M and the tree T_ω to the remaining numbers.

Theorem 5.5. *For $M \subseteq \mathbb{N}$, we have $\mathfrak{A}_M \equiv^{\mathrm{w}}_k T_\omega$ for all $k \in \mathbb{N}$, and M can be reduced to the* FO*-theory of the infinitary Muchnik-iteration* $(\mathfrak{A}^\infty_M, \mathrm{cl})$.

Proof. Using Ehrenfeucht-Fraïssé-games and Prop. 5.2, one obtains

$$\mathfrak{A}_M \equiv^{\mathrm{w}}_k (b^*T_\omega, \preceq, \varepsilon) \cong T_\infty \equiv^{\mathrm{w}}_k T_\omega \ .$$

For the second statement, it suffices, by Lemma 5.4, to reduce M to the first-order theory of $(A_M, \preceq, \varepsilon, U)$ where $U = b^* \cup \{b^m \mid m \in M\}a^*$ is the set of nodes of the tree $\mathfrak{A}_M$ that belong to some infinite branch. ∎

If a transformation t is $(\mathrm{FO}, \mathrm{MSO}^{\mathrm{w}})$-compatible, then for any structure $\mathfrak{A}$, the FO-theory of $t(\mathfrak{A})$ can be reduced to the the $\mathrm{MSO}^{\mathrm{w}}$-theory of $\mathfrak{A}$. Contrary to this, the above theorem states that the FO-theory of the infinitary Muchnik-iteration can be arbitrarily more complicated than the $\mathrm{MSO}^{\mathrm{w}}$-theory of the base structure. Hence we obtain

Corollary 5.6. *Infinitary Muchnik-iteration is not* $(\mathrm{FO}, \mathrm{MSO}^{\mathrm{w}})$*-compatible.*

6. Summary

Table 1 summarizes our knowledge about the compatibility of Muchnik's and Shelah & Stupp's iteration. It consists of four subtables dealing with finitary and infinitary Muchnik-iteration and with finitary and infinitary Shelah-Stupp-iteration. The sign + in cell $(\mathcal{K}, \mathcal{L})$ of a subtable denotes that the respective iteration is $(\mathcal{K}, \mathcal{L})$-compatible, – denotes the opposite. Minus-signs without further marking hold since the base structure can be defined in any of its iterations. Capital letters denote references: (A) is [15], (B) [17], (C) [10, Prop. 3.4], (D) [10, Thm. 4.10], (E) Theorem 4.3, (F) Theorem 5.5, and (G) since the base structure is definable in its iteration and finiteness of a set is no MSO-property. Small letters denote that the result follows from Theorem 6.1 below and some further "simple" arguments from the result marked by the corresponding capital letter.

Theorem 6.1. *Let* $(\mathcal{K}, \mathcal{L})$ *be any of the pairs of logics* $(\mathrm{MSO}^{\mathrm{closed}}, \mathrm{MSO})$, $(\mathrm{MSO}^{\mathrm{ch}}, \mathrm{MSO}^{\mathrm{ch}})$, *or* $(\mathrm{MSO}^{\mathrm{mch}}, \mathrm{MSO}^{\mathrm{mch}})$. *There exists a computable function* red *such that, for any* σ*-structure* $\mathfrak{A}$, red *reduces the* $\mathcal{K}$*-theory of* $(\mathfrak{A}^\infty, \mathrm{cl})$ *to the* $\mathcal{L}$*-theory of* $(\mathfrak{A}^*, \mathrm{cl})$.

The same holds for the Shelah-Stupp-iterations.

The two questions marks in Table 1 express that it is not clear whether finitary Muchnik-iteration is $\mathrm{MSO}^{\mathrm{w}}$-compatible or not.

Note the main difference between Muchnik- and Shelah-Stupp-iteration: the latter is $\mathcal{K}$-compatible for all relevant logics while only MSO behaves that nicely with respect to (infinitary) Muchnik-iteration

A referee proposed to also consider the variant of MSO where set quantification is restricted to countable sets. As to whether Muchnik iteration is compatible with this logic is not clear at the moment.

	Muchnik				inf. Muchnik		
	MSO	MSO^{w}	FO		MSO	MSO^{w}	FO
MSO	+ (B)	–	–	$\text{MSO}^{\text{closed}}$	+ (b)	–	–
MSO^{w}	– (g)	?	–	MSO^{w}	– (g)	– (f)	–
FO	+ (b)	?	– (C)	FO	+ (b)	– (F)	– (c)

	Shelah-Stupp				inf. Shelah-Stupp		
	MSO	MSO^{w}	FO		MSO	MSO^{w}	FO
MSO	+ (A)	–	–	$\text{MSO}^{\text{closed}}$	+ (a)	–	–
MSO^{mch}	– (g)	+ (E)	–	MSO^{mch}	– (g)	+ (e)	–
MSO^{w}	– (G)	+ (e)	–	MSO^{w}	– (g)	+ (e)	–
MSO^{ch}	+ (a)	+ (e)	+ (D)	MSO^{ch}	+ (a)	+ (e)	+ (d)
FO	+ (a)	+ (e)	+ (d)	FO	+ (a)	+ (e)	+ (d)

Table 1: summary

References

[1] A. Blumensath and S. Kreutzer. An extension of Muchnik's theorem. *Journal of Logic and Computation*, 15:59–64, 2005.

[2] D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS'02*, Lecture Notes in Comp. Science vol. 2420, pages 165–176. Springer, 2002.

[3] Th. Colcombet and Ch. Löding. Transforming structures by set interpretations. *Logical Methods in Computer Science*, 3:1–36, 2007.

[4] B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126:53–75, 1994.

[5] B. Courcelle and I. Walukiewicz. Monadic second-order logic, graph coverings and unfoldings of transition systems. *Ann. Pure Appl. Logic*, 92(1):35–62, 1998.

[6] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1991.

[7] S. Feferman and R.L. Vaught. The first order properties of algebraic systems. *Fund. Math.*, 47:57–103, 1959.

[8] Y. Gurevich and S. Shelah. Rabin's uniformization problem. *J. of Symb. Logic*, 48:1105–1119, 1983.

[9] D. Kuske and M. Lohrey. Logical aspects of Cayley-graphs: The monoid case. *International Journal of Algebra and Computation*, 16:307–340, 2006.

[10] D. Kuske and M. Lohrey. Monadic chain logic over iterations and applications to push-down systems. In *LICS 2006*, pages 91–100. IEEE Computer Society, 2006.

[11] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[12] M.O. Rabin. *Automata on infinite objects and Church's problem*. American Mathematical Society, Providence, R.I., 1972. Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics, No. 13.

[13] D. Seese. The structure of models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic*, 53:169–195, 1991.

[14] A.L. Semenov. Decidability of monadic theories. In M. Chytil and V. Koubek, editors, *MFCS'84*, Lecture Notes in Comp. Science vol. 176, pages 162–175. Springer, 1984.

[15] S. Shelah. The monadic theory of order. *Annals of Mathematics*, 102:379–419, 1975.

[16] W. Thomas. On chain logic, path logic, and first-order logic over infinite trees. In *LICS'87*, pages 245–256. IEEE Computer Society Press, 1987.

[17] I. Walukiewicz. Monadic second-order logic on tree-like structures. *Theoretical Computer Science*, 275:311–346, 2002.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 479-490
www.stacs-conf.org

GEOMETRIC SET COVER AND HITTING SETS FOR POLYTOPES IN $\mathbb{R}^3$

SÖREN LAUE

Max-Planck-Institut für Informatik, Campus E1 4, 66123 Saarbrücken, Germany
E-mail address: soeren@mpi-inf.mpg.de

Abstract. Suppose we are given a finite set of points P in $\mathbb{R}^3$ and a collection of polytopes $\mathcal{T}$ that are all translates of the same polytope T. We consider two problems in this paper. The first is the set cover problem where we want to select a minimal number of polytopes from the collection $\mathcal{T}$ such that their union covers all input points P. The second problem that we consider is finding a hitting set for the set of polytopes $\mathcal{T}$, that is, we want to select a minimal number of points from the input points P such that every given polytope is hit by at least one point.

We give the first constant-factor approximation algorithms for both problems. We achieve this by providing an epsilon-net for translates of a polytope in $\mathbb{R}^3$ of size $O(\frac{1}{\epsilon})$.

Introduction

Suppose we are given a set of n points P in $\mathbb{R}^3$ and a collection of polytopes $\mathcal{T}$ that are all translates of the same polytope T. We consider two problems in this paper. The first is the set cover problem where we want to select a minimal number of polytopes from the collection $\mathcal{T}$ such that their union covers all input points P. The second problem that we consider is finding a hitting set for the set of polytopes $\mathcal{T}$, that is, we want to select a minimal number of points from the input points P such that every given polytope is hit by at least one point.

Both problems, the set cover problem and the hitting set problem which are in fact dual to each other are very fundamental problems and have been studied intensively. In a more general setting, where the sets could be arbitrary subsets, both problems are known to be NP-hard, in fact they are even hard to approximate within $o(\log n)$ [11]. Even when the sets are induced by geometric objects it is widely believed that the corresponding set cover problem as well as the hitting set problem are NP-hard. Several geometric versions of these problems were even proven to be hard to approximate. Hence, we are looking for algorithms that approximate both problems. We give the first constant-factor approximation algorithms for the set cover problem and the hitting set problem for translates of a polytope in $\mathbb{R}^3$. The central idea to our approximation algorithms are small *epsilon-nets*.

1998 ACM Subject Classification: F.2.2, G.2.1.

Key words and phrases: Computational Geometry, Epsilon-Nets, Set Cover, Hitting Sets.

This work was supported by the Max Planck Center for Visual Computing and Communication (MPC-VCC) funded by the German Federal Ministry of Education and Research (FKZ 01IMC01).

A set of elements P (also called points) along with a collection $\mathcal{T}$ of subsets of P (also called ranges) is in general called a *set system* $(P, \mathcal{T})$ and for geometric settings also known as *range spaces*. One essential characteristic of these set systems is the *Vapnik-Chervonenkis dimension*, or *VC-dimension* [17]. The VC-dimension is the cardinality of the largest subset $A \subseteq P$ for which $\{T \cap A : T \in \mathcal{T}\}$ is the powerset of A. If the set A is finite, we say that the set system $(P, \mathcal{T})$ has bounded VC-dimension, otherwise we say the VC-dimension of $(P, \mathcal{T})$ is unbounded. For instance, the set system induced by translates of a polytope has VC-dimension three as well as the set system induced by halfspaces in $\mathbb{R}^2$. A set $N \subseteq P$ is called an *epsilon-net* for a given set system $(P, \mathcal{T})$ if $N \cap T \neq \emptyset$ for every subset $T \in \mathcal{T}$ for which $\|T\| \geq \epsilon \cdot \|P\|$. In other words, an epsilon-net is a hitting set for all subsets $T \in \mathcal{T}$ whose cardinality is an ϵ-fraction of the cardinality of the input point set P.

It is known that there exist epsilon-nets of size $O\left(\frac{d}{\epsilon} \log \frac{d}{\epsilon}\right)$ for any set system of VC-dimension d [2, 10]. This bound is in fact tight for arbitrary set systems as there exist set systems that do not admit epsilon-nets of size less than this bound [16]. Such an epsilon-net can be simply found by random sampling [12].

However, for special set systems that are induced by geometric objects there do exist epsilon-nets of smaller size, namely of size $O(\frac{1}{\epsilon})$. It has been shown by Pach and Woeginger [16] that halfspaces in $\mathbb{R}^2$ and translates of polytopes in $\mathbb{R}^2$ admit epsilon-net of size $O(\frac{1}{\epsilon})$. Matoušek et al. [14] gave an algorithm for computing small epsilon-nets for pseudo-disks in $\mathbb{R}^2$ and halfspaces in $\mathbb{R}^3$. The result for halfspaces in $\mathbb{R}^3$ also follows from a more general statement by Matoušek [13].

Among other reasons for finding epsilon-nets of small size is the fact that an epsilon-net of size $g(\epsilon)$ immediately implies an approximation algorithm for the corresponding hitting set with approximation guarantee of $O(g(1/c)/c)$, where c denotes the optimal solution to the hitting set [15]. This means, that for arbitrary set systems of fixed VC-dimension we have an algorithm for the hitting set problem with approximation $O(\log c)$. And for set systems that admit epsilon-nets of size $O(1/\epsilon)$ we get an approximation algorithm to the hitting set problem with constant approximation guarantee.

Clarkson and Varadarajan [5] developed a technique that connects the complexity of a union of geometric objects to the size of the epsilon-net for the dual set system. Using this result, they are able to develop, among other approximation algorithms for geometric objects in $\mathbb{R}^2$, a constant-factor approximation algorithm for the set cover problem induced by translates of unit cubes in $\mathbb{R}^3$.

We extend their result to not only the set cover problem but also the hitting set problem for arbitrary translates of a polytope in $\mathbb{R}^3$. We do not require the polytope to be convex or fat. This is the first constant-factor approximation algorithm for these two problems. We achieve this by giving an epsilon-net for translates of a polytope in $\mathbb{R}^3$ of size $O(\frac{1}{\epsilon})$. We reduce the problem of finding epsilon-nets for translates of a polytope to a family of non-piercing objects in $\mathbb{R}^2$ and then generalize the epsilon-net finder for pseudo-disks of Matoušek et al. [14] to our setting.

The set cover problem which is studied by Hochbaum and Maass [9] where one is allowed to move the objects is fundamentally different. They give a PTAS for their problem.

1. Small Epsilon-Nets for Polytopes in $\mathbb{R}^3$

Let P be a set of n points in $\mathbb{R}^3$ and let $\mathcal{T}$ be a family of polytopes that are all translates of the same bounded polytope T_0. We want to find a set of polytopes of minimal cardinality

among the collection $\mathcal{T}$ that covers all input points P. First, we find a small epsilon-net for this set system and use this later for the constant-factor approximation of the hitting set problem. Finally, we show how this then can be translated into a solution for the set cover problem.

Throughout this paper we denote by T the polytope as well as the subset of points from P that T covers and by $\mathcal{T}$ the family of polytopes as well as the corresponding family of subsets of P. This will make the paper easier to read and it will be clear from the context whether we talk about the geometric object or the corresponding set of points.

1.1. From Polytopes in $\mathbb{R}^3$ to Non-Piercing Objects in $\mathbb{R}^2$

So given such a set system $(P, \mathcal{T})$ we want to find an epsilon-net for it, i.e. we are looking for a set $N \subseteq P$ such that every subset of points $T \in \mathcal{T}$ with $\|T\| \geq \epsilon \cdot \|P\|$ is stabbed by at least one point from N.

We can cut the polytope T into, lets say k polytopes $T_1, T_2, \ldots, T_k$. If the polytope T contains ϵn input points then one of the polytopes $T_1, T_2, \ldots, T_k$ must contain at least $\frac{\epsilon}{k} \cdot n$ input points. Hence, in order to find an ϵ-net for the set system $(P, \mathcal{T})$ induced by translates of T, it suffices to find $\frac{\epsilon}{k}$-net for the set systems induced by the translates of $T_1, T_2, \ldots, T_k$.

Following this reasoning we can reduce our problem for finding an epsilon-net for the set system induced by translates of arbitrary polytopes to translates of *convex* polytopes by cutting the possibly non-convex polytope into a set of convex polytopes. Note that the number of these convex polytopes only depends on the polytope T and hence is constant for fixed T.

Wlog. let T be from now on a convex polytope. We can place a cubical grid onto the space $\mathbb{R}^3$ such that for any translate of T every cubical grid cell contains at most vertex of T. This can be achieved by making the grid fine enough. Clearly, the maximal number t of grid cells that can be intersected by T is bounded and only depends on T. Again, if T contains ϵn input points then at least one of the cells must contain at least $\frac{\epsilon}{t} \cdot n$ of the input points. Hence, we can restrict ourselves to finding epsilon-nets for translates of triangular cones where all input points lie in a cube in $\mathbb{R}^3$. This just adds a multiplicative constant to the size of the final epsilon-net.

The case when the cubical cell only contains a halfspace or the intersection of two halfspaces can be either seen as a special case of a cone or, in fact, be even treated separately in a much simpler way. The case of a translate of a halfspace reduces to a one-dimensional problem an admits an epsilon-net of size 1 and the case of two intersecting halfspaces reduces to a problem on intervals which admits an epsilon-net of size $O(1/\epsilon)$.

In the following we will construct an epsilon-net for the set system $(P, \mathcal{C})$ that is induced by translates of a triangular cone C.

Given a cone C, we call a set of points P in non-C-degenerate position if every translate of C has at most three points of P on its boundary. We can always perturb the input points P in such a way that they are in non-C-degenerate position and the collection of subsets of the form $P \cap C_T$ where C_T is a translate of C does not decrease [6]. Hence, we can restrict ourselves on non-C-degenerate set of points P.

We place a coordinate system such that the input points all have z-coordinate greater than 0 and a ray r emitting from the apex of the cone C and lying entirely in the cone should intersect the plane $z = 0$. We refer to such a cone as a cone that *opens to the bottom*

and the ray r as its *internal ray*. Figure 1 illustrates this setup for the two-dimensional case.

The following two definitions are helpful generalizations the lower envelope.

Definition 1.1. Given a finite point set P and a triangular cone C that opens to the bottom consider the arrangement of all translates of C that have a point of P on its boundary but no point of P in its interior. The upper set of plane segments that can be seen from above is called the *lower envelope of P with respect to cone C*.

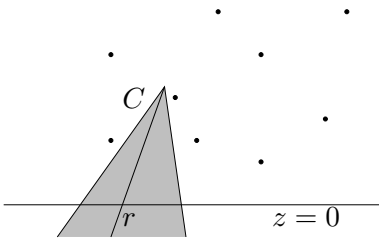


Figure 1: The cone C and its internal ray r.

Figure 2 illustrates the definition of the lower envelope in the two-dimensional case. This definition is similar to the definition of alpha-shapes where the cone is replaced by a ball. We call all points that lie on the lower envelope with respect to cone C *lower envelope points* and denote this set by L.

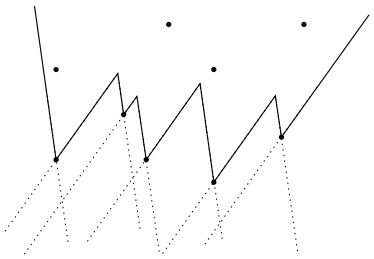


Figure 2: The lower envelope with respect to cone C, the corresponding cones are drawn dotted.

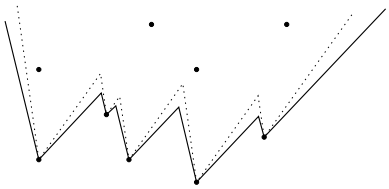


Figure 3: The flattened lower envelope with respect to cone C, lower envelope is drawn dotted.

Definition 1.2. Let C be a triangular cone that opens to the bottom and let $P \subseteq \mathbb{R}^3$ be a finite set of points in non-$\mathcal{C}$-degenerate position. Let C' be a cone that is flatter that C by small δ and such that it contains C and the combinatorial structure of P and C' is the same as for P and C. See figure 3 for an illustration. Then, the lower envelope of P with respect to C' is called the *flattened lower envelope of P with respect to cone C*.

Such a cone C' always exists for a finite point set that is in non-$\mathcal{C}$-degenerate position. From now on we will abbreviate the term lower envelope with respect to cone C by lower envelope since we will throughout this paper only talk about the same cone C. The flattened lower envelope can be basically seen as a slightly flattened version of the lower envelope.

The next lemma shows that we can reduce the problem of finding an epsilon-net with respect to cones of arbitrary point sets to lower envelope points.

Lemma 1.3. *If for every finite point set $P' \subseteq \mathbb{R}^3$ of lower envelope points in non-C-degenerate position there exists an epsilon-net with respect to translates of a cone C of size $s(\epsilon)$ then there exists an epsilon-net with respect to translates of a cone C of size $3s(\epsilon)$ for every finite point set $P \subseteq \mathbb{R}^3$ in non-C-degenerate position.*

Proof. Let $P \subseteq \mathbb{R}^3$ be such a finite point set in non-$\mathcal{C}$-degenerate position and let C denote the cone. Let L denote the set of lower envelope points. Let $\bar{L} = P \setminus L$ be the set of all non-lower envelope points. We project all non-lower envelope points $\bar{L}$ along the internal ray r of cone C onto the flattened lower envelope (cf. figure 4). We denote the projection of a point p by p'. Let P' be union of the projected points and L. Clearly, P' is a set of lower envelope points in non-C-degenerate position.

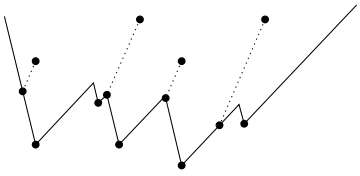


Figure 4: The projection of points onto flattened lower envelope.

Suppose we have an epsilon-net N' for this point set P'. From this epsilon-net N' we will construct an epsilon-net N for the original point set P. If a point from the set L is in the epsilon-net N', we also add it to the epsilon-net N for P. If however, a projected point p' is in N' then we add to N the three points p_1, p_2 and p_3 from the lower envelope L that determine the cone C on whose boundary also p' lies. Note that whenever an arbitrary cone contains the point p' then it has to contain one of the three points p_1, p_2 or p_3.

We have the following two properties:

(1) If a cone contains at least ϵn points from the set P then it contains at least ϵn points from the set P'.
(2) If a cone contains a point from the epsilon-net N' for P' then the cone contains a point from the epsilon-net N for P.

Both properties prove that the set N is indeed an epsilon-net for P. ∎

The preceding lemma assures that we can restrict ourselves on a finite set of lower envelope points in non-C-degenerate position. For such a set system we will now construct a corresponding set system of points in the plane and a collection of regions in the plane.

Definition 1.4. Let C be a cone and let P' be a finite set of lower envelope points in non-C-degenerate position and let $\mathcal{C}$ be a collection of translates of C. We define a projection τ from the flattened lower envelope onto the plane $z = 0$ by projecting each point along the internal ray r. Let the projection of all points $p' \in P'$ which all lie on the be denoted as the set S. For each cone of the collection the image of the intersection of the cone with the flattened lower envelope is an object $D \subseteq \mathbb{R}^2$ and the family $\mathcal{C}$ of cones induces a family of objects which we will denote by $\mathcal{D}$.

Using the flattened lower envelope instead of the lower envelope avoids degeneracy. The intersection of an arbitrary cone with the flattened lower envelope is always a collection of line segments. Furthermore, it makes everything continuous in the sense that if a cone is moved continuously in $\mathbb{R}^3$ then the intersection of the cone with the flattened lower envelope moves continuously as well as its image of the projection τ. Note, that τ is injective.

Analogously, we call a set of points $S \subseteq \mathbb{R}^2$ in non-$\mathcal{D}$-degenerate position if every $D \in \mathcal{D}$ has at most three points on its boundary. We have the following lemma:

Lemma 1.5. *If for every finite point set $S \subseteq \mathbb{R}^2$ in non-$\mathcal{D}$-degenerate position there exists an epsilon-net with respect to the family of objects $\mathcal{D}$ produced by the projection τ of size $s(\epsilon)$ then there exists an epsilon-net with respect to cones of size $s(\epsilon)$ for every point set of lower envelope points $P' \subseteq \mathbb{R}^3$ in non-$\mathcal{C}$-degenerate position.*

Proof. The proof follows easily from the fact that the image of a cone C under the projection τ contains exactly those points that are the image of the points that are contained in C. ∎

We refer to a cone C as the corresponding cone of the object $D = \tau(C)$. We will prove a few useful properties of the so constructed set system $(S, \mathcal{D})$.

Notice, that the intersection of two triangular cones is again a cone. Furthermore, the intersection of a possibly infinite family of triangular cones is either empty or again a triangular cone since all cones are closed. The intersection of the boundary of a cone with the flattened lower envelope is either empty or a set of line segments that form one simple closed cycle. Hence, the image of a cone under the projection τ is a closed and connected region whose boundary is a closed and connected cycle.

Definition 1.6. Two geometric objects(sets) $A \subseteq \mathbb{R}^2$ and $B \subseteq \mathbb{R}^2$ that are bounded by Jordan curves are said to be *non-piercing* if the boundary of A and B cross at most twice. A family of geometric objects is called non-piercing if every two objects from this family are non-piercing. See figure 5 for an illustration.

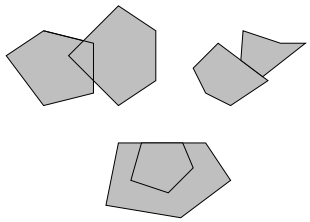


Figure 5: A set of non-piercing objects

Lemma 1.7. *The projection τ produces a family $\mathcal{D}$ of non-piercing objects.*

Proof. Consider two cones C_1 and C_2 that intersect each other. If one is contained in the other, i.e. $C_1 \subseteq C_2$ then we are done, as $\tau(C_1) \subseteq \tau(C_2)$ and hence their boundaries cannot cross. So if C_1 and C_2 intersect and none is subset of the other then the intersection of their boundaries are two rays emitting from the same point. Each of these rays intersects the flattened lower envelope exactly once. Hence, as the projection τ is injective the boundary of the two images of the cones C_1 and C_2 under the projection τ intersect exactly twice. Thus, the objects are non-piercing. ∎

1.2. Small Epsilon-Nets for Non-Piercing Objects in $\mathbb{R}^2$

In this subsection we will derive a few properties of the projection that are necessary to apply the algorithm of Matoušek et al. [14] for finding a small epsilon-net for pseudo-disks. These properties also hold in general for any family of non-piercing objects with the additional property that for any three points there always exists an object that has these three points on its boundary. However the proofs are a bit more involved. Since this does not lie in the scope of this paper, we omit this here and focus only on the special family of non-piercing objects that is produced by the projection described above.

Consider the family of all cones that have p and q on its boundary. The intersection of all these cones is a cone C_{pq} that has p and q on its boundary. Connect p and q by a Jordan curve E_{pq} such that it lies entirely in the cone C_{pq} and on the flattened lower envelope, for instance part of the boundary of C_{pq} that intersects the flattened lower envelope. The image of E_{pq} under the projection τ is a curve $\tau(E_{pq})$ embedded in the plane.

Definition 1.8. Let $\mathcal{D}$ be a family of non-piercing objects and let $S \subseteq \mathbb{R}^2$ be a finite set of points. We call two points $p, q \in \mathbb{R}^2$ *$\mathcal{D}$-Delaunay neighbors* if there exists an object $D \in \mathcal{D}$ that has p and q on its boundary and no other point of S in its interior. The $\mathcal{D}$-Delaunay graph of S, in short $\mathcal{D}$-DT(S), is the graph that is embedded in the plane, has S as its vertex set and the edges $\tau(E_{pq})$ between all $\mathcal{D}$-Delaunay neighbors p and q.

Due to the definition of the $\mathcal{D}$-Delaunay edge between two $\mathcal{D}$-Delaunay neighbors p and q it is guaranteed that whenever a object $D \in \mathcal{D}$ contains p as well as q then it also must contain the $\mathcal{D}$-Delaunay edge $\tau(E_{pq})$. In the following we will prove that this $\mathcal{D}$-Delaunay graph is in fact a triangulation of the vertex set S.

Lemma 1.9. *The $\mathcal{D}$-Delaunay graph of the given finite point set S in non-$\mathcal{D}$-degenerate position is a triangulation.*

Proof. First, we will prove that $\mathcal{D}$-DT(S) is planar. Suppose otherwise, i.e. two edges $\tau(E_{pq})$ and $\tau(E_{rs})$ intersect each other in the plane. Since the cone C_{pq} does not have any point in its interior and C_{rs} also does not have any point in its interior and since each of these cones has at most 3 points on its boundary the objects $\tau(C_{pq})$ and $\tau(C_{rs})$ would have to pierce each other, see figure 6 for an illustration. Here, it is actually essential, that the set S is in non-$\mathcal{D}$-degenerate position. Thus, the graph is planar.

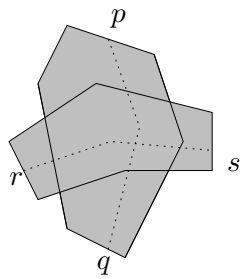


Figure 6: Two intersecting $\mathcal{D}$-Delaunay edges and their defining objects

The graph $\mathcal{D}$-DT(S) itself consists of an outer face which is defined by cones of the lower envelope that have at most 2 points on their boundary and all other faces are triangles defined by the cones of the lower envelope that have exactly three points on its boundary. Suppose an inner face F is not bounded by a triangle. Then, one can place the apex of a cone in such a way onto the flattened lower envelope such that its image under the projection τ is a point which lies inside this face F. By moving the cone upward one can ensure that the cone will finally have three points on its boundary whose image under the projection τ are three vertices of the face F but no point in its interior. Hence, the face F must be bounded by a triangle. Hence, $\mathcal{D}$-DT(S) is a triangulation of the set S. ∎

We call the points of S that lie define the outer face the *convex hull of S with respect to cone C* and we denote it by $\text{conv}_C(S)$. It is a generalization of the standard convex hull and we will make use of it later. For a standard triangulation one requires that the outer face is determined by the convex hull. Here, we replaced the standard convex hull by the convex hull with respect to cone C. This is the appropriate generalization that we need.

Lemma 1.10. *Let D be an object produced by the projection τ. The subgraph G of $\mathcal{D}$-DT(S) induced by the points of S that lie in D is connected.*

Proof. We prove the connectivity using induction over the number of points that lie in D. If D contains at most 2 points that it must be connected by definition and the fact that we can slide down the corresponding cone until both points lie on the boundary. So lets assume that every object D that contains at most k points from the set S induces a connected subgraph G. Now consider an object D that contains $k+1$ points of S. Consider the cone that is the intersection of all cones that contain exactly those $k+1$ points. This cone has exactly three points on its boundary. We can move the cone by a small δ in such a way that each of the three points can be excluded separately. As all of these induced graphs are connected by induction hypothesis, the whole subgraph induced by D must be connected. ∎

We need two more lemmas. Both lemmas basically rely on the fact that projection τ is continuous.

Lemma 1.11. *Let S be a finite point set.*

(1) *For any object $D \in \mathcal{D}$, there exists an object $D' \in \mathcal{D}$ with $S \cap D' = S \cap \operatorname{int} D' = S \cap D$.*
(2) *For any object $D' \in \mathcal{D}$, there exists an object $D \in \mathcal{D}$ with $S \cap D' = S \cap \operatorname{int} D' = S \cap \operatorname{int} D$.*

Proof. Let C be the corresponding cone of D. If we move C upward along the internal ray r by a small δ then the corresponding object D' of this cone will satisfy (1). On the other hand, if we move the cone C downward along the ray r by a small δ then the corresponding object D' will satisfy (2). ∎

Lemma 1.12. *Let S be a finite point set in non-$\mathcal{D}$-degenerate position, let (p, q) be a $\mathcal{D}$-Delaunay edge in $\mathcal{D}$-DT(S). Then, there exists an object D with p and q on its boundary and with $S \cap D = \{p, q\}$.*

Proof. Let D be the object that assures that p, q is a $\mathcal{D}$-Delaunay edge, i.e. D has p and q on its boundary. Since the point set S is in non-$\mathcal{D}$-degenerate position D has at most three points on its boundary. If D has exactly two points on its boundary we are done. So lets assume that D has exactly three points on its boundary. Let C be the corresponding cone of D and let the corresponding points of p and q be $p' \in \mathbb{R}^3$ and $q' \in \mathbb{R}^3$. Neither p' nor q' can lie on the intersection of two of the defining planes of cone C because otherwise the cone could still be moved in an upward direction such that all three points still lie on the boundary until the cone hits a fourth point. But this would mean that the point set was in $\mathcal{C}$-degenerate position. Hence, p' and q' lie in the interior of two of the plane segments of cone C. If we now move the cone C downward by a small δ such that it still touches p' and q' then the corresponding object of this cone will only have p and q on its boundary. ∎

Having these properties, we can basically directly apply the algorithm for finding an epsilon-net for pseudo-disks from [14]. We will describe the algorithm here and prove its correctness for our setting.

We are given a finite point set S in non-$\mathcal{D}$-degenerate position and we want to find a subset $N \subseteq S$ of size $O(1/\epsilon)$ that stabs any object D that contains at least ϵn points of S.

Let $\delta = \epsilon/6$. First, let $S_1, \ldots, S_j$ be pairwise disjoint subsets of S with the following properties: Each S_i contains δn points, their union contains the convex hull of S with respect to cone C, i.e. $\operatorname{conv}_C(S) \subseteq \bigcup_{1 \le i \le j} S_i$ and each S_i is representable by $S \cap \tau(C_i)$ for an appropriate cone C_i. Such sets can be easily constructed by repeatedly biting off points from $\operatorname{conv}_C(S)$ with a suitable cone C_i. Notice, that all these objects $D_i = \tau(C_i)$ belong to the collection $\mathcal{D}$.

Next, find a maximal pairwise disjoint collection $S_{j+1}, \ldots, S_k$ of subsets of the remaining points $S \setminus \bigcup_{1 \le i \le j} S_i$ satisfying $S_i = S \cap D_i$ for some object D_i and each subset containing δn points. Obviously, there are at most $1/\delta + 1$ many subsets S_i in total. For an illustration we refer to figure 7. We assign all points in S_i the color i and call all other points *colorless*. Let $\bar{S}$ be the set of all colored points. Note, that if an object contains only colorless points then it contains less that δn points, since the collection of subsets S_i was maximal.

Let G be the $\mathcal{D}$-Delaunay graph of the set of colored points $\bar{S}$, i.e. $G = \text{DT}(\bar{S})$. G is indeed a triangulation (cf. lemma 1.9). In this graph we call a triangle *uni-colored*, *bi-colored* or *tri-colored* depending upon the number of colors its vertices have. In a similar way we call edges uni-colored or bi-colored. We call a maximal connected chain of bi-colored triangles in G sharing bi-colored edges a *corridor* (cf. figure 8). Since the graph G is planar and each of the induced subgraphs $G \cap D_i$ is connected according to lemma 1.10 the number of such corridors is at most $3k - 6$ ([14]). All colorless points are contained in the corridors and

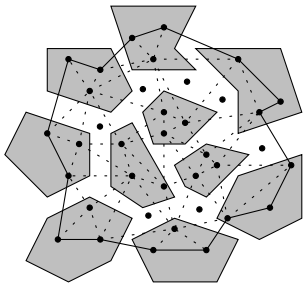


Figure 7: The sets S_i and the convex hull $\text{conv}_C(S)$ with respect to cone C. The $\mathcal{D}$-Delaunay triangulation is drawn dotted.

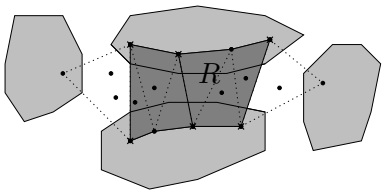


Figure 8: The corridor R which is split into two sub-corridors and two tri-colored triangles. The corners of the sub-corridors are marked by crosses.

the tri-colored triangles because any uni-colored triangle is contained it its color-defining object. We break each corridor R into a minimum number of *sub-corridors*, i.e. sub-chains of the chain that forms R, so that each sub-corridor contains at most δn colorless points. Since there are less than n colorless points and since the total number of corridors is $3k-6$ the total number of sub-corridors is $O(1/\delta)$.

Each sub-corridor is bounded by two chains of uni-colored edges which we call *sides* and by two bi-colored edges which we call *ends* of the sub-corridor. The endpoints of the sides are called *corners*. Let $N \subseteq S$ be the set of all corners of all sub-corridors. Since each sub-corridor has at most 4 corners the size of N is $O(1/\epsilon)$. The set N is an epsilon-net for the set of non-piercing objects $\mathcal{D}$.

The proof that N is indeed an epsilon-net relies in principle on the fact that the collection $\mathcal{D}$ are non-piercing objects and follows along the lines of [14].

Proof. Let D be an object that has no points of S on its boundary (cf. lemma 1.11) and assume that D does not contain any points from N. The theorem is proven when we can show that D then contains less than ϵn points of S. If D contains no colored point then we are done, because the sets S_i were a maximal. Hence, D must contain at least one colored point. If it contains two colored points, lets say z_1 of color 1 and z_2 of color 2, we can draw the following picture: Let D_1 be the color defining object of color 1 and D_2 the color defining object of color 2. Then D intersects D_1 and D_2 but cannot pierce them. The area between D_1 and D_2 is a sub-corridor whose ends we denote by (a_1, a_2) and (b_1, b_2). Lemma 1.12 assures that there is an object D_a that has a_1 and a_2 on its boundary and there is an object D_b that has b_1 and b_2 on its boundary. Since D also does not contain any point from N which are the corners of the sub-corridors, i.e. it does not contain a_1, a_2, b_1 or b_2 and since D and D_a as well as D and D_b are non-piercing it must lie between two ends of one sub-corridor. See figure 9 for an illustration. Now, as all objects D_1, D_2, D_a and D_b contain at most δn points and the sub-corridor also contains at most δn points D can contain at most $5 \cdot \delta n = 5/6\epsilon n < \epsilon n$ points of S.

The case where D only contains points of one color and colorless points is very similar. There is basically only one setup and it is depicted in figure 10. Arguing as above it easy to see in this case that D cannot contain more than $4 \cdot \delta n < \epsilon n$ points from S. ∎

Hence, we have the following theorem

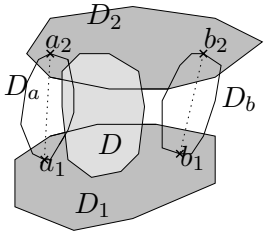


Figure 9: The case where D contains colored points of at least two colors.

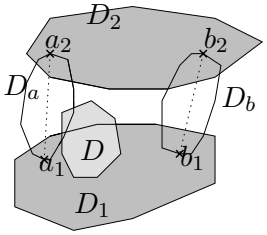


Figure 10: The case where D contains colored points of exactly one color.

Theorem 1.13. *Let $\mathcal{D}$ be the set of non-piercing objects in $\mathbb{R}^2$, that is produced by the projection τ. For every finite point set in non-$\mathcal{D}$-degenerate position there exists an epsilon-net of size $O(1/\epsilon)$.*

Together with lemma 1.3 and lemma 1.5 this immediately implies our main theorem

Theorem 1.14. *Given a finite point set $P \subseteq \mathbb{R}^3$ and a polytope $T \subseteq \mathbb{R}^3$. The set system $(P, \mathcal{T})$ induced by a set of translates of polytope T admits an epsilon-net of size $O(1/\epsilon)$.*

2. From Epsilon-Nets to Hitting Sets

In this section we will describe a constant factor approximation algorithm to the hitting set problem using the epsilon-net of size $O(1/\epsilon)$ from the previous section. Recall that in the hitting set problem we are given a set of points $P \in \mathbb{R}^3$ and a set polytopes that are all translates of the same polytope and we would like to select a subset $H \subseteq P$ of the input points of minimal cardinality such that every polytope is stabbed by a point in H. We denote the corresponding set system by $(P, \mathcal{T})$. The fractional hitting set problem is a relaxation of the original hitting set problem and is defined by the following linear program:

$$\min \quad \textstyle\sum_{p \in P} x(p) \tag{2.1}$$

$$\text{s.t.} \quad \forall T \in \mathcal{T} \quad \sum_{p \in T} x(p) \geq 1 \tag{2.2}$$

$$\forall p \in P \quad x(p) \geq 0 \tag{2.3}$$

Let OPT denote the optimal size of the hitting set and OPT^* the optimal value of the fractional hitting set problem. It is known that the integrality gap is constant for set systems that admit an epsilon-net of size $O(1/\epsilon)$ [15].

Let $w : P \to \mathbb{R}_{\geq 0}$ be a weight function for the set P. We define the weight $w(A)$ of a subset $A \subseteq P$ to be the sum of the weights of the elements of A. The weighted version of an epsilon-net is as follows:

Definition 2.1. Consider a set system $(P, \mathcal{T})$ and a weight function $w : P \to \mathbb{R}_{\geq 0}$. A set $H \subseteq P$ is called an *epsilon-net with respect to w* if $H \cap R \neq \emptyset$ for every subset $T \in \mathcal{T}$ for which $w(T) \geq \epsilon \cdot w(S)$.

There are algorithms that compute a hitting set provided one has an epsilon-net finder. The core idea to all these algorithms is to find a weight function $w : P \to \mathbb{R}_{\geq 0}$ that assigns weights to the elements of P and finds an appropriate ϵ such that every set in $\mathcal{T}$ has weight

at least $\epsilon \cdot w(S)$. Once such weights are found it is then obvious that an epsilon-net to this set system is automatically a hitting set.

The algorithm given by Brönnimann and Godrich [3] computes these weights iteratively. Initially, all elements have weight 1. Then, in each iteration an epsilon-net is computed and then checked whether it is also a proper hitting set. If not, i.e. there is a set which is not hit, then the weights of its elements are doubled. This is done until a hitting set is found. This algorithm can be seen as a deterministic analogue of the randomized natural selection technique used for instance by Clarkson [4].

Another algorithm is by Even et al. [7]. Here, the weights of the elements and ϵ are directly found by the following linear program:

$$\max \epsilon \tag{2.4}$$

$$\text{s.t.} \quad \forall T \in \mathcal{T} \quad w(T) \geq \epsilon \tag{2.5}$$

$$\sum_{p \in P} w(p) = 1 \tag{2.6}$$

$$\forall p \in P \quad w(p) \geq 0 \tag{2.7}$$

It suffices to approximate the solution to this linear problem. There are numerous algorithms that find an approximate solution to such a covering linear program efficiently [18, 8].

One can reduce the problem of finding a weighted epsilon-net to the unweighted case. One just makes multiple copies of a point according to its assigned weight and it can be shown that the cardinality of this multiset can be bounded by $2n$ [5]. Hence, an $\frac{\epsilon}{2}$-net for this set system gives a hitting set for the original hitting set problem. Hence, we have

Theorem 2.2. *There exists a polynomial time algorithm that computes a constant-factor approximation to the hitting set problem for translates of polytopes in* $\mathbb{R}^3$.

3. From Hitting Set to Set Cover

Definition 3.1. The *dual set system* of a set system $(P, \mathcal{T})$ is the set system $(\mathcal{T}, P^*)$ where $P^* = \{\mathcal{T}_p : p \in P\}$ and $\mathcal{T}_p$ consists of all subsets of $\mathcal{T}$ that contain p.

Obviously, a set cover for the primal set system is a hitting set for the dual set system. Hence, in order to solve the set cover problem for a set system it suffices to solve the hitting set problem for the dual set system. For arbitrary set systems, the dual set system can be of quite different structure. In general it is only known that the VC-dimension of the dual set system is less than 2^{d+1}, where d is the VC-dimension of the primal set system [1].

However, we observe that if the set system is induced by translates of a polytope, then the dual is again induced by translates of a polytope. To see this, let $(P, \mathcal{T})$ be the primal set system. One just reduces each polytope $T \in \mathcal{T}$ to a point, for instance each to its lowest vertex. Let this be the set P'. Then, replace each point of P by a translate of the polytope T' which is the inversion of T in a point. One easily verifies that the so constructed set system $(P', \mathcal{T}')$ of points P' and collection of translates of polytope T' is indeed equivalent to the dual $(\mathcal{T}, P^*)$. This holds in fact for all $\mathbb{R}^d$. Hence, we can find a constant-factor approximation to the set cover problem for translates of a polytope in $\mathbb{R}^3$ in polynomial time. This brings us to our final theorem

Theorem 3.2. *There exists a polynomial time algorithm that computes a constant-factor approximation to the set cover problem for translates of polytopes in* $\mathbb{R}^3$.

4. Conclusions and Open Problems

In this paper we have given the first constant-factor approximation algorithm for finding a set cover for a set of points in $\mathbb{R}^3$ by a given collection of translates of a polytope as well as the first constant-factor approximation algorithm for the corresponding hitting set problem. We achieved this result by providing an epsilon-net of size $O(\frac{1}{\epsilon})$ for the corresponding set system which is optimal up to a multiplicative constant. Eventhough we can approximate a unit ball in $\mathbb{R}^3$ up to any given precision by a polytope, the corresponding question, whether there exists a constant-factor approximation algorithm for unit balls in $\mathbb{R}^3$ still remains open.

Acknowledgements

The author would like to thank Nabil H. Mustafa and Saurabh Ray for useful discussion on the topic and an anonymous referee for pointing out an error in a preliminary version of this paper.

References

[1] P. Assouad. Densité et dimension. *Ann. Inst. Fourier*, 33(3):233–282, 1983.
[2] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the vapnik-chervonenkis dimension. *J. ACM*, 36(4):929–965, 1989.
[3] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite vc-dimension: (preliminary version). In *SoCG '94*, pages 293–302, New York, NY, USA, 1994. ACM Press.
[4] K. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995.
[5] K. L. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. In *SoCG '05*, pages 135–141, New York, NY, USA, 2005. ACM Press.
[6] H. Edelsbrunner and E. Welzl. On the number of line separations of a finite set in the plane. *J. Comb. Theory, Ser. A*, 38(1):15–29, 1985.
[7] G. Even, D. Rawitz, and S. Shahar. Hitting sets when the vc-dimension is small. *Inf. Process. Lett.*, 95(2):358–362, 2005.
[8] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS '98*, page 300, Washington, DC, USA, 1998. IEEE Computer Society.
[9] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, 1985.
[10] J. Komlós, J. Pach, and G. J. Woeginger. Almost tight bounds for epsilon-nets. *Discrete and Computational Geometry*, 7:163–173, 1992.
[11] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.
[12] J. Matousek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.
[13] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2(3):169–186, 1992.
[14] J. Matoušek, R. Seidel, and E. Welzl. How to net a lot with little: Small epsilon-nets for disks and halfspaces. In *SoCG '90*, pages 16–22, 1990.
[15] J. Pach and P. K. Agarwal. *Combinatorial Geometry*. Wiley, New York, 1995.
[16] J. Pach and G. Woeginger. Some new bounds for epsilon-nets. In *SoCG '90*, pages 10–15, New York, USA, 1990. ACM Press.
[17] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probability. *Theory Probab. Appl.*, 16:264–280, 1971.
[18] N. E. Young. Randomized rounding without solving the linear program. In *SODA '95*, pages 170–178, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 491-502
www.stacs-conf.org

A THEORY FOR VALIANT'S MATCHCIRCUITS (EXTENDED ABSTRACT)

ANGSHENG LI [1] AND MINGJI XIA [1]

[1] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences,
P.O. Box 8717, Beijing 100080, China
E-mail address: angsheng@ios.ac.cn
E-mail address: xmjljx@gmail.com

Abstract. The computational function of a matchgate is represented by its character matrix. In this article, we show that all nonsingular character matrices are closed under matrix inverse operation, so that for every k, the nonsingular character matrices of k-bit matchgates form a group, extending the recent work of Cai and Choudhary [1] of the same result for the case of $k = 2$, and that the single and the two-bit matchgates are universal for matchcircuits, answering a question of Valiant [4].

1. Introduction

Valiant [4] introduced the notion of matchgate and matchcircuit as a new model of computation to simulate quantum circuits, and successfully realized a significant part of quantum circuits by using this new model. Valiant's new method organizes certain computations based on the graph theoretic notion of perfect matching and the corresponding algebraic object of the Pfaffian. This leaves an interesting open question of characterizing the exact power of the matchcircuits. To solve these problems, a significant first step would be a better understanding the structures of the matchgates and the matchcircuits, to which the present paper is devoted.

In [6], Valiant introduced the notion of *holographic algorithm*, based on matchgates and their properties, but with some additional ingredients of the choice of a set of linear basis vectors, through which the computation is expressed and interpreted.

Matchgates and their character matrices have some nice properties, which have already been extensively studied. In [1], Cai and Choudhary showed that a matrix is the character matrix of a matchgate if and only if it satisfies all the useful Grassmann-Plücker identities, and all nonsingular character matrices of two bits matchgates form a group.

1998 ACM Subject Classification: F.1.1.

Key words and phrases: Pfaffian,Matchgate, Matchcircuit.

Both authors are supported by the NSFC Grant no. 60325206 and no. 60310213. The second author is also supported by MATHLOGAPS (MEST-CT-2004-504029).

In the present paper, we show that for every k, all the nonsingular character matrices of k-bit matchgates form a group, extending the result of Cai and Choudhary of the same result for the case of $k = 2$.

Furthermore, we show that every matchcircuit based on k-bit matchgates for $k > 2$ can be realized by a series of compositions of either single bit or two bits matchgates. This result answers a question raised by Valiant in [4]. The result is an analogy of the quantum circuits in the matchcircuits [3].

We organize the paper as follows. In section 2, we outline necessary definitions and background of the topic. In section 3, we state our results, and give some overview of the proofs. In section 4, we establish our first result that for every k, all nonsingular k-bit character matrices form a group. In section 5, we prove the second result that level 2 matchgates are universal for matchcircuits.

2. Definitions

2.1. Graphs and Pfaffian

Let $G = (V, E, W)$ be a weighted undirected graph, where $V = \{1, 2, \ldots, n\}$ is the set of vertices each represented by a distinct positive integer, E is the set of edges and W is the set of weights of the edges. We represent the graph by a skew-symmetric matrix M, called the *skew-symmetric adjacency matrix* of G, where $M(i, j) = w(i, j)$ if $i < j$, $M(i, j) = -w(i, j)$ if $i > j$, and $M(i, i) = 0$.

The *Pfaffian* of an $n \times n$ skew-symmetric matrix M is defined to be 0 if n is odd, 1 if n is 0, and if $n = 2k$ where $k > 0$ then it is defined by

$$\mathrm{P}f(M) = \sum_{\pi} \epsilon_\pi w(i_1, i_2)w(i_3, i_4) \ldots w(i_{2k-1}, i_{2k}),$$

where

- $\pi = [i_1, i_2, \ldots, i_{2k}]$ is a permutation on $[1, 2, \ldots, n]$,
- the summation is over all permutations π, where $i_1 < i_2, i_3 < i_4, \ldots, i_{2k-1} < i_{2k}$ and $i_1 < i_3 < \ldots < i_{2k-1}$,
- ϵ_π is the sign of the permutation π, or equivalently, ϵ_π is the sign or parity of the number of overlapping pairs, where a pair of edges $(i_{2r-1}, i_{2r}), (i_{2s-1}, i_{2s})$ is *overlapping* iff $i_{2r-1} < i_{2s-1} < i_{2r} < i_{2s}$ or $i_{2s-1} < i_{2r-1} < i_{2s} < i_{2r}$.

A *matching* is a subset of edges such that no two edges share a common vertex. A vertex is said to be *saturated* if there is a matching edge incident to it. A *perfect matching* is a matching which saturates all vertices. There is a one-to-one correspondence between the monomials in the Pfaffian and the perfect matchings in G.

If M is an $n \times n$ matrix and $A = \{i_i, \ldots, i_r\} \subseteq \{1, \ldots, n\}$, then $M[A]$ denotes the matrix obtained from M by deleting the rows and columns of indices in A. The *Pfaffian Sum* of M is a polynomial over indeterminates $\lambda_1, \lambda_2, \ldots, \lambda_n$ defined by

$$\mathrm{PfS}(M) = \sum_{A} (\prod_{i \in A} \lambda_i) \mathrm{Pf}(M[A])$$

where the summation is over the 2^n subsets of $\{1, \ldots, n\}$. There is a one-one correspondence between the terms of the Pfaffian sum and the matchings in G. We consider only instances such that each λ_i is fixed to be 0 or 1. In this case, Pfaffian Sum is a summation over all

matchings that match all nodes with $\lambda_i = 0$. It is well known that both the Pfaffian and the Pfaffian Sum are computable in polynomial time.

2.2. Matchgate

A *matchgate* Γ, is a quadruple (G, X, Y, T), where $G = (V, E, W)$ is a graph, $X \subseteq V$ is a set of input nodes, $Y \subseteq V$ is a set of output nodes, and $T \subseteq V$ is a set of *omittable nodes* such that X, Y and T are pairwise disjoint. Usually the numbers of nodes in V are consecutive from 1 to $n = |V|$ and X, Y have minimal and maximal numbers respectively. Whenever we refer to the Pfaffian Sum of a matchgate fragment, we assume that $\lambda_i = 1$, if $i \in T$, and 0 otherwise. Each node in $X \cup Y$ is assumed to have exactly one incident *external edge*. For a node in X, the other end of the external edge is assumed to have index less than the index for any node in V, and for a node in Y, the other end node has index greater than that for every node in V. If $k = |X| = |Y|$, then Γ is called *k-bit matchgate*. A matchgate is called a *level k matchgate*, if it is an n-bit matchgate for some $n \leq k$. If a matchgate only contains input nodes, output nodes and one ommitable node, then it is called a *standard matchgate*.

We define, for every $Z \subseteq X \cup Y$, the *character $\chi(\Gamma, Z)$ of Γ with respect to Z* to be the value $\mu(\Gamma, Z)\mathrm{PfS}(G - Z)$, where $G - Z$ is the graph obtained from G by deleting the vertices in Z together with their incident edges, and the *modifier* $\mu(\Gamma, Z) \in \{-1, 1\}$ counts the parity of the number of overlaps between matched edges in $G - Z$ and matched external edges. We assume that all the nodes in Z are matched externally. By definition of the modifier, it is easy to verify that $\mu(\Gamma, Z) = \mu(\Gamma, Z \cap X)\mu(\Gamma, Z \cap Y)$, and that if $X = \{1, 2, \ldots, k\}$ and $Z \cap X = \{i_1, i_2, \ldots, i_l\}$, then $\mu(\Gamma, Z \cap X) = (-1)^{\sum_{j=1}^{l}(i_j - j)}$.

The *character matrix* $\chi(\Gamma)$ is defined to be the $2^{|X|} \times 2^{|Y|}$ matrix such that entry $(i_1 i_2 \ldots i_k, i_n i_{n-1} \ldots i_{n-k+1})$ is $\chi(\Gamma, X' \cup Y')$, where $X' = \{j \in X | i_j = 1\}$, $Y' = \{j \in Y | i_j = 1\}$ and $i_1 i_2 \ldots i_k$, $i_n i_{n-1} \ldots i_{n-k+1}$ are binary expression of numbers between 0 and $2^k - 1$. We also use (X', Y') to denote this entry. We call an entry (X', Y') *edge entry*, if $0 < |(X - X') \cup (Y - Y')| \leq 2$. Throughout the paper, we identify a matchgate and its character matrix. An easy but useful fact is that for every k, the $2^k \times 2^k$ unit matrix is a character matrix.

2.3. Properties of character matrix

We introduce several properties of character matrices, which will be used in the proof of our results.

Theorem 2.1 ([4]). *If A and B are character matrices of size $2^k \times 2^k$, then AB is a character matrix.*

Theorem 2.2 ([4]). *Given any matchgate Γ there exists another matchgate Γ' that has the same character as Γ and has an even number of nodes, exactly one of which is omittable.*

Theorem 2.3 ([1]). *Let A be a $2^k \times 2^l$ matrix. Then A is the character matrix of a k-input, l-output matchgate, if and only if A satisfies all the useful Grassmann-Plücker identities.*

This is a very useful characterization of the character matrices generalizing the characterization for a major part of all 2-input 2-output matchgates in [4]. The proof of this theorem implies the following:

Corollary 2.4 ([1]). *Let A be a $2^k \times 2^l$ matrix whose right-bottom most entry is* 1 *satisfying all the useful Grassmann-Plücker identities. Then A is uniquely determined by its edge entries and A is the character matrix of a standard matchgate Γ containing $k+l+1$ nodes (k input nodes, l output nodes and 1 omittable node).*

Recently, Cai and Choudhary also showed that:

Theorem 2.5 ([1]). *Let A be a 4×4 character matrix. If A is invertible, then A^{-1} is a character matrix. Consequently, the nonsingular 4×4 character matrices form a group.*

2.4. Matchcircuit

Given a matchgate $\Gamma = (G, X, Y, T)$, we say that it is *even*, if $\mathrm{PfS}(G - Z)$ is zero whenever $Z = X \cup Y$ has odd size, and *odd* if $\mathrm{PfS}(G - Z)$ is zero whenever $|Z|$ is even.

Theorem 2.6 ([4],[1]). *Consider a matchcircuit Γ composed of gates as in* [4]. *Suppose that every gate is:*

(1) *a gate with diagonal character matrix,*
(2) *an even gate applied to consecutive bits $x_i, x_{i+1}, \ldots, x_{i+j}$ for some j,*
(3) *an odd gate applied to consecutive bits $x_i, x_{i+1}, \ldots, x_{i+j}$ for some j, or*
(4) *an arbitrary gate on bits $x_1, \ldots, x_j$ for some j.*

Suppose also that every parallel edge above any odd matchgate, if any, has weight -1 and all other parallel edges have weight 1. Then the character matrix of Γ is the product of the character matrices of the constituent matchgates, each extended to as many inputs as those of Γ.

From now on, whenever we say a matchcircuit, we mean that it satisfying the requirements in the above theorem. An example circuit is shown in Fig. 1, where the edges in a matchgate are not drown, and each node has index smaller than that of all nodes located to the right of the node. We call a matchcircuit is of *level k*, if it is composed of matchgates no more than k bits.

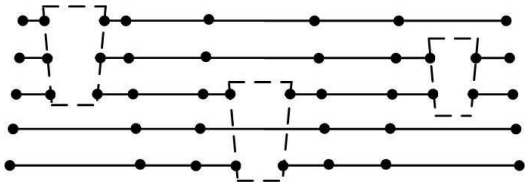


Figure 1: An example of matchcircuit.

The *character matrix of a matchcircuit* is defined by the same way as that of a matchgate except that there is no modifier μ.

3. The results and overview of the proofs

Theorem 3.1. *For every k, the nonsingular $2^k \times 2^k$ character matrices form a group under the matrix multiplication.*

We will prove theorem 3.1 by induction on the size of matchgates. The proof proceeds as follows. Based on corollary 2.4, we observe that all $2^k \times 2^k$ character matrices can be transformed to a special form $2^k \times 2^k$ character matrices. This suggests the following:

Definition 3.2. We say that a k-bit matchgate is a *reducible matchgate*, if the bottom pair of nodes k and $n-k+1$ are connected by a weight 1 edge, and there is no other edge incident to any of the nodes k and $n-k+1$.

The character matrix of a reducible matchgate is called a *reducible character matrix*.

By corollary 2.4, a character matrix B is a reducible character matrix if it satisfies the following:

(1) $B_{2^k-1,2^k-1} = B_{2^k-2,2^k-2} = 1$.
(2) All the edge entries in the last two columns and the last two rows are 0 except for $B_{2^k-2,2^k-2}$.

Firstly we prove that if the k-bit nonsingular character matrices are closed under matrix inverse operation, then so are the $(k+1)$-bit nonsingular reducible character matrices .

Secondly, we introduce some elementary nonsingular matchgates so that every nonsingular $2^k \times 2^k$ character matrix can be transformed to a reducible character matrix by multiplying with the character matrices of the elementary matchgates.

This transformation is realized by four phases as follows. Starting from $A = A^{(0)}$, we need the following:

Phase T1 ($A^{(0)} \Rightarrow A^{(1)}$). Turn the right-bottom most entry to 1.

Phase T2 ($A^{(1)} \Rightarrow A^{(2)}$). Turn the edge entries in the last row and column to 0's, while keeping the right-bottom most entry 1.

Phase T3 ($A^{(2)} \Rightarrow A^{(3)}$). Turn the entry $A^{(2)}_{2^k-2,2^k-2}$ to 1, while keeping the right-bottom most entry 1 and the edge entries in the last row and column 0's.

Phase T4 ($A^{(3)} \Rightarrow A^{(4)}$). Turn the edge entries in the row 2^k-2 and column 2^k-2 to 0's, while keeping the last two diagonal entries 1's and the edge entries in the last row and column 0's.

Each phase consists of several *actions* (or for simplicity, steps). In each step, either the positions of entries are changed, or the values of some entries are changed.

An action is defined to be the multiplication of a character matrix with an elementary character matrix. The role of an action is to change some specific entries to be some fixed value 0 or 1. However, such an action will certainly injure other entries which are undesired.

The crucial observation is that an appreciate sequence of actions will gradually satisfy all the entries requirements. During the course of the transformation, once an entry requirement is satisfied by some action, it will never be injured again by the future actions. That is to say, an action may injure only the entries which have not been satisfied. This ensures that all the entries requirements will be eventually satisfied.

This describes the idea of the proof of theorem 3.1. The proof will also build an essential ingredient for our second result, the theorem below.

Theorem 3.3. *For every $k > 2$, if Γ is a matchcircuit composed of level k matchgates, then:*

(1) Γ *can be simulated by a level* 2 *matchcircuit* Δ.
(2) *A* k*-bit matchgate can be simulated by* $O(k^4)$ *many single and two-bit matchgates. And every matchcircuit* Γ *can be simulated by a level* 2 *matchcircuit in polynomial time.*

Our proof of theorem 3.3 is a composition of the proof of theorem 3.1 and some more elementary matchgates. On the other hand, one could firstly prove theorem 3.3, then prove 3.1 by combining theorem 3.3 and theorem 2.5. However there are subtle difference between character matrices of matchgate and matchcircuit. Therefore, this approach needs additional technique.

4. Group property of the k-bit character matrices

In this section, we prove theorem 3.1. To proceed an inductive argument, we exploit the structure of the reducible character matrices which pave the way to the reductions.

4.1. Reducible matchgates

Lemma 4.1. *Let* Δ_1 *be a* $(k+1)$*-bit reducible matchgate, that is, the bottom edge* $(k+1, k+3)$ *having weight* 1 *and there is no any other edge incident to any of the nodes* $k+1$ *and* $k+3$. *Let* Γ_1 *be the* k*-bit matchgate obtained from* Δ_1 *by deleting the edge* $(k+1, k+3)$. *Then:*
(i) If Δ_1 *is invertible, so is* Γ_1.
(ii) If $\chi(\Gamma_1)^{-1}$ *is a character matrix, so is* $\chi(\Delta_1)^{-1}$.

Proof. (Sketch) For (i). This holds because $\chi(\Delta_1)$ is a block diagonal matrix after rearranging the order of rows and columns, and $\chi(\Gamma_1)$ is equal to one block.

For (ii). We prove this by constructing the inverted matchgate $\Delta_2(F_2, W_2, Z_2, T_2)$ of $\Delta_1(F_1, W_1, Z_1, T_1)$ from the inverted gate $\Gamma_2(G_2, X_2, Y_2, T_2)$ of $\Gamma_1(G_1, X_1, Y_1, T_1)$.

It suffices to prove that the composition of Δ_1 and Δ_2 has the unit matrix as its character matrix. See Fig. 2 for the intuition of the proof, while detailed verification will be given in the full version of the paper.

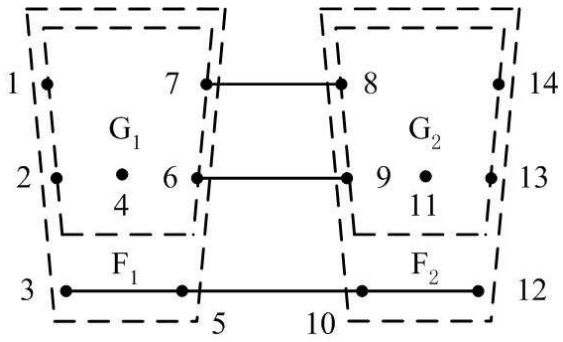


Figure 2: Example of $k = 2$. $X_1 = \{1, 2\}$, $Y_1 = \{6, 7\}$, $T_1 = \{4\}$ $X_2 = \{8, 9\}$, $Y_2 = \{13, 14\}$, $T_2 = \{11\}$, $W_1 = \{1, 2, 3\}$, $Z_1 = \{5, 6, 7\}$, $W_2 = \{7, 8, 9\}$, $Z_2 = \{12, 13, 14\}$.

■

4.2. The transformation lemma

In this part, we construct the matchgates to realize the phases T1 – T4 prescribed in section 3, and show that every k-bit nonsingular character matrix can be transformed to a k-bit reducible character matrix by using the transformation.

The key point to the proof of the theorem is the following:

Lemma 4.2. *Let A be a $2^k \times 2^k$ nonsingular character matrix. Then there exist nonsingular character matrices $L_s, \ldots, L_2, L_1, R_1, R_2, \ldots, R_t$ for some s and t such that $L_s \cdots L_2 L_1 A R_1 R_2 \cdots R_t$ is a reducible character matrix.*

Proof. Given a nonsingular character matrix A, we denote A by $A^{(0)}$. We construct the matchgates to realize the four phases T1 – T4. We use $A^{(i)}$ to denote the character matrix obtained from $A^{(i-1)}$ by using phase Ti, where $i = 1, 2, 3, 4$. We start with $A^{(0)}$, and define the transformation to be a series of actions, defined in section 3. In the discussion below, we will use A to denote the character matrix obtained so far in the construction from $A^{(0)}$ (or shortly, the current matrix).

The four phases proceed as follows.

Phase T1: Suppose that Γ_l is the k-bit matchgate such that the l-th pair of input-output nodes are connected by a path of length 2 on which each edge has weight 1, and each of the other pairs is connected by an edge of weight 1, and $k+1$ is the only unomittable node other than the input and output nodes. (See Fig. 3 (a)). Let C_l denote the character matrix of Γ_l.

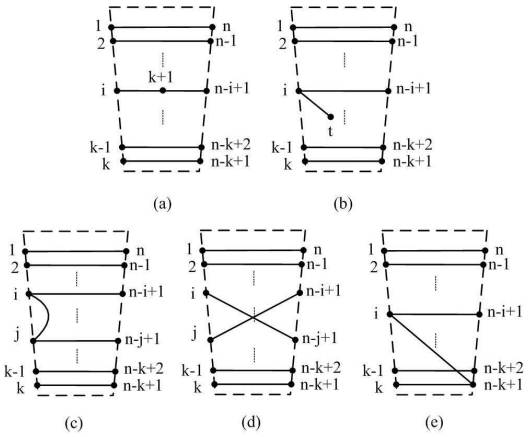


Figure 3:

Suppose without loss of the generality that $A_{I=i_1 i_2 \ldots i_k, J=j_1 j_2 \ldots j_k} \neq 0$. Define

$$L_1 = \prod_{1 \leq l \leq k, i_l = 0} C_l, \quad R_1 = \prod_{1 \leq l \leq k, j_l = 0} C_l.$$

It is easy to see that the right-bottom most entry, a say, of $L_1 A R_1$ is either $A_{I,J}$ or $-A_{I,J}$ by computing the $\mathrm{PfS}(G - X \cup Y)$ of the composed matchgate corresponding to $L_1 A R_1$. Let $L_2 = \frac{1}{a} E$, and $A^{(1)} = L_2 L_1 A R_1$. Clearly L_2 is a character matrix, so is $A^{(1)}$, by theorem 2.1.

Phase T2: Phase T2 will change the edge entries in the last column and the last row to 0's. We first describe the actions for the column as follows.

Phase T2 for the last column:

We turn the edge entries in the last column to 0's one by one from bottom to top. To turn an edge entry $(X', 2^k - 1)$ to zero, we need a row transformation applied to the current matrix A, which adds the multiplication of $-b$ and the last row to row X', where b is the value of entry $(X', 2^k - 1)$ of the current matrix.

Therefore phase T2 for the last column consists of the following actions. In decreasing order of X', for every edge entry $(X', 2^k - 1)$, we have:

Action $(X', 2^k - 1)$: Multiplying an elementary character matrix, L say, to the current character matrix A from the left side, where L is a character matrix satisfying that the diagonal entries are all 1's, and that $L_{I=X',2^k-1} = -b$.

This makes some row transformations to the current matrix according to the nonzero entries other than the diagonal entries. The row transformation corresponding to $L_{I=X',2^k-1} = -b$ is exactly the one that realizes the goal of this action.

Now we formally construct the matchgate to realize the character matrix L as required in the action $(X', 2^k - 1)$ above. The construction is divided into two cases depending on the size of X' as follows.

Case 1. $X' = X - \{i\}$ for some i.

We use the matchgate with the following properties: (1) each input-output pair of the gate is connected by an edge of weight 1, and (2) it contains one more edge (i, t) to realize L, where t is the unique omittable node, and the weight of (i, t) is either b or $-b$ ensuring $L_{I,2^k-1} = -b$. For intuition of the matchgate, a reader is referred to Fig. 3 (b).

Let (I', J') be an arbitrary nonzero entry of L other than the diagonal entries. By the construction of the gate, we have that the i-th bit of I' and J' are 0 and 1 respectively, and that I', and J' are identical on the j-th bit for every $j \neq i$. Hence $I' < J'$ and $I' \leq I$ (recall that $I = X'$). The action at entry $(X', 2^k - 1)$ in this case actually makes the following row transformation: For each such pair (I', J'), row I' is added by the multiplication of $L_{I',J'}$ and row J'. Since $I' \leq I$, all the edge entries $(I_1, 2^k - 1)$ with $I_1 > I$ have never been injured by the action in this case.

Case 2 $X' = X - \{i, j\}$ for some i, j.

The character matrix L in this case is constructed by a similar way to that in case 1 above, using the matchgate in Fig. 3 (c).

The cost of the action in this case is similarly analyzed to that for case 1.

Recall that after phase T1, the right-bottom most entry is 1. The actions in both case 1 and case 2 of phase T2 above will never injure the last row of the matrix, so that the satisfaction of T1 is still preserved by the current state of the construction.

Phase T2 for the last row: The construction, and analysis for the actions is the same as that for the column case with the roles of rows and columns exchanged.

Therefore, the goal of T2 prescribed in section 3 has been realized.

Phase T3: The goal of this phase is similar to that of phase T1, but different actions are needed. T3 consists of 2 actions. The first action moves a nonzero edge entry to position $(2^k - 1, 2^k - 1)$, and the second one changes edge entry $(2^k - 1, 2^k - 1)$ to 1. The actions proceed as follows.

Action 1: First, we choose a nonzero edge entry. Since $A^{(2)}$ is nonsingular, there must be a nonzero edge entry $A^{(2)}_{X'=X-\{i\},Y'=Y-\{j\}}$ for some i and j. (Otherwise, all edge entries are zero's so that $A^{(2)}$ is a zero matrix, contradicting the non-singularity of $A^{(0)}$.)

We use a gate of type Γ_d, defined as follows: (i) connect each input-output pair other than the i-th or the j-th pair by an edge, (ii) the i-th input is connected to the j-th output, and (iii) the j-th input is connected to the i-th output. All edges are of weight 1. (See Fig. 3 (d)) Let $C_{i,j}$ denote the character matrix of the matchgate described above.

This action just turns $A^{(2)}$ to $C_{i,k}A^{(2)}C_{j,k}$ by connecting the matchgate of $C_{i,k}$ with the gate of $A^{(2)}$, and the gate of $C_{i,k}$ in the order of left to right.

Firstly, we verify that action 1 realizes its goal. Generally, multiplying $C_{a,b}$ from left (resp. right) side is equivalent to exchanging pairs of rows (resp. columns) $i_1 i_2 \dots i_a \dots i_b \dots i_k$ and $i_1 i_2 \dots i_b \dots i_a \dots i_k$, modular a factor of 1 or -1. Hence, the edge entry $(2^k-2, 2^k-2)$ of $C_{i,k}A^{(2)}C_{j,k}$ is either $A^{(2)}_{X',Y'}$ or $-A^{(2)}_{X',Y'}$.

Secondly, we analyze the cost of the action. Notice that the row exchanges are determined by a bit exchange on the labels of rows, so that the number of zeros in (the string of) the row label is kept unchanged. By definition, an edge entry can be exchanged only with another edge entry. Therefore all edge entries in the last row and column are kept zeros. In addition, it is easy to see that the left-bottom most entry is kept 1.

Action 2: We construct a matchgate with all of the input-output pairs connected by an edge of weight 1, except that the last pair is connected by an edge of weight $w = \frac{1}{A_{2^k-2,2^k-2}}$.

All entries of the character matrix of this matchgate are zeros, except for the diagonal entries. A diagonal entry (I, I) is w, if the last bit of I is 0, and 1, otherwise.

We multiply this character matrix with the current matrix, then a straightforward calculation shows that entry $(2^k-1, 2^k-1)$ is turned to 1, while all the satisfied entries achieved previously are still preserved.

The goal of T3 is realized.

Phase T4: This phase is similar to phase T2, except that we need consider the consequence on the last column and row. We start from changing the edge entries in column 2^k-2.

Phase T4 for column 2^k-2: Suppose we are going to change edge entry $(X-\{i\}, Y-\{n-k+1\})$ to zero by the order from bottom to top. Denote the action realizing this goal by *action at* $(X-\{i\}, Y-\{n-k+1\})$.

We construct the elementary matchgate used in the action at $(X-\{i\}, Y-\{n-k+1\})$. Each pair of input-output nodes of this matchgate is connected by an edge of weight 1, furthermore, the i-th input node is connected to the last output node by an edge of weight w, where w is either $A_{X-\{i\},Y-\{n-k+1\}}$ or $-A_{X-\{i\},Y-\{n-k+1\}}$ such that entry $(X-\{i\}, Y-\{n-k+1\})$ of the character matrix of the matchgate is $-A_{X-\{i\},Y-\{n-k+1\}}$. (See Fig. 3 (e).)

We examine the nonzero entries in the character matrix L of the constructed matchgate. We first note that all diagonal entries are 1's. Let (I', J') denote an arbitrary nonzero entry other than the diagonal entries of the matrix L. By construction of the matchgate, I' and J' differ at only the i-th and the k-th bits, and $I'|_i = J'|_k = 0$, $I'|_k = J'|_i = 1$, $I' < J'$, $I' < X-\{i\}$ and I', J' contain the same number of 0's, which is at least 1, where $I'|_i$ denotes the i-th bit of I'. The action at $(X-\{i\}, Y-\{n-k+1\})$ multiplies L with A from the left side. It makes some row transformations: for every such entry (I', J') chosen as above, add row I' by the multiplication of row J' by $L_{I',J'}$. So the goal of this action is realized.

Now we analyze the cost of the action. We first prove that it does not injure the edge entries in column $2^k - 2$ which have already been satisfied. The reason is similar to that in phase T2. Because $I' \leq X - \{i\}$, the action only injures the rows with indices less than $X - \{i\}$.

The cost of the action is different from that in phase T2 in that it may affect the edge entries in the last column which have already been satisfied in phases T1 and T2. Because I' and J' contain the same number of 0's, which is at least 1, all the row changes made by the action always add a zero edge entry of the last column to another zero edge entry in the same column. Hence, it does not injure the satisfied entries in the last column. Additionally, it is obvious that the last two rows are preserved during the current action, so the left-bottom most entry, the edge entries in the last row and entry $(2^k - 2, 2^k - 2)$ are all preserved.

Phase T4 for row $2^k - 2$: Similar actions to that in phase T4 for the column above can be applied to the row $2^k - 2$ to change its edge entries to 0's.

Therefore, T4 realizes its goal, at the same time, it preserves the satisfied entries in phases T1 – T3.

We have realized the phases T1 – T4 prescribed in section 3, by corollary 2.4, B is a reducible character matrix. The lemma follows. ∎

4.3. Proof of theorem 3.1

Proof. We prove by induction on k that for every k, and every $2^k \times 2^k$ character matrix A, if A is invertible, then A^{-1} is a character matrix.

The case for $k = 1$ is easy, the first proof was given by Valiant in [4].

Suppose by induction that the theorem holds for $k - 1$. By lemma 4.2, there exist nonsingular character matrices L_i and R_j such that $B = L_s \cdots L_2 L_1 A R_1 R_2 \cdots R_t$ is the character matrix of a reducible matchgate Δ. Let B' be the $2^{k-1} \times 2^{k-1}$ character matrix of Γ constructed from Δ by deleting the bottom edge.

Since A is invertible, so is B, and so is B' by lemma 4.1. By the inductive hypothesis, B'^{-1} is a character matrix, so is B^{-1} by lemma 4.1.

By the choice of L_i and R_j, for all $1 \leq i \leq s$ and $1 \leq j \leq t$, we have that

$$A^{-1} = R_1 R_2 \cdots R_t B^{-1} L_s \cdots L_2 L_1.$$

By theorem 2.1, A^{-1} is also a character matrix.

This completes the proof of theorem 3.1. ∎

We notice that the inductive argument in the proof of theorem 3.1 also gives a different proof for the result in the case of $k = 2$. Our method is a constructive, and uniform one. It may have some more applications.

5. Level 2 matchgates are universal

We introduce nine types of matchgates as our elementary gates. We use $\Gamma_a, \ldots, \Gamma_i$, to denote the elementary level 2 matchgates corresponding to that in the following Fig. 4 (a), (b), (c), (d), (e), (f), (g), (h) and (i) respectively.

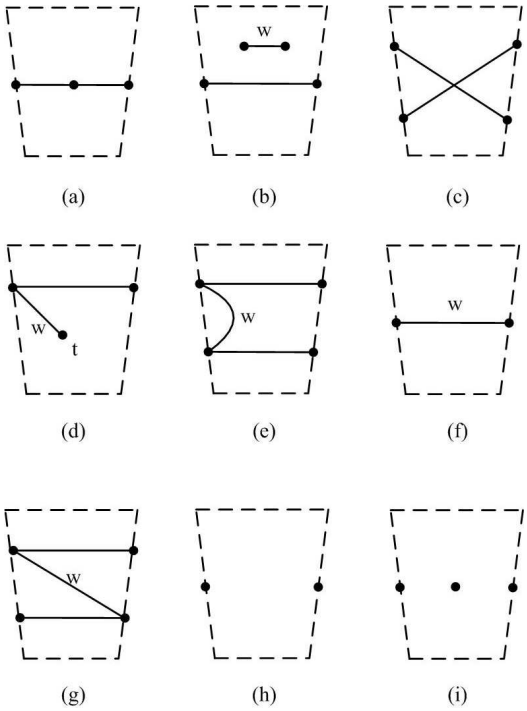


Figure 4:

We describe the elementary gates as follows. All edges in Γ_a have weight 1. All edges connecting an input and an output node except for the edge in Γ_f, and the diagonal edge in Γ_g, are all of weight 1. The remaining edges take weights w.

Γ_a makes a row (or column, when it is multiplied from right side) exchange, which is a special transformation, of the character matrix according to a bit flip on the label, and it is used to move a nonzero entry to the right-bottom most entry by the same way as that in the proof of theorem 2.3 in [1]. Γ_b is used to realize cE, and to turn a nonzero entry to 1. Both Γ_a and Γ_b are only used in the first phase, i.e. T1, of the transformation. Intuitively, Γ_c can exchange two consecutive bits, and it allows us to apply some other elementary gates to nonconsecutive bits. Γ_d and Γ_e are used in phase T2 to eliminate the edge entries in the last column and the last row. Γ_c will be also used in phase T3 to move a nonzero edge entry to position $(2^k-2, 2^k-2)$, in which case, Γ_f will further turn this entry to 1. Γ_g is used in phase T4 to eliminate the edge entries in the column 2^k-2 and row 2^k-2. A nonzero singular character matrix will be transformed to a matchcircuit composed of only Γ_h-type gates. Γ_i is used to realize zero matrix. To understand the composition of Γ_c with other elementary gates, we need the following:

Lemma 5.1. *Suppose A is the character matrix of a k-bit matchcircuit Δ, and P_1, P_2 are two arbitrary permutations on k elements. There exists matchcircuit Λ constructed from Δ by adding some gates Γ_c, such that the corresponding character matrices B satisfying $B_{2^k-1,2^k-1} = A_{2^k-1,2^k-1}$ and $|B_{i_1\cdots i_k,j_1\cdots j_k}| = |A_{P_1(i_1\cdots i_k),P_2(j_1\cdots j_k)}|$.*

The following lemma gives the transformation for matchcircuits.

Lemma 5.2. *For any $k > 2$, and any k-bit matchcircuit Δ consisting of a single nonsingular k-bit matchgate Γ, there is a new matchcircuit Λ constructed by adding some invertible single and two-bit matchgates to Δ, such that the character matrix B of Λ is reducible. Furthermore, B is the character matrix of an even reducible matchgate.*

So far we have established the result for the first significant case that a matchgate is applied to the first k bits.

In the following lemma we consider two more cases:

- a gate applied to consecutive bits but not starting from the first bit,
- a gate applied to nonconsecutive bits.

For the first case, the gate must be an even or an odd gate, we observe that only even and odd gates are used in the transformation for an even or an odd gate. For the second case, we extend its matrix, and replace it by a new even gate which is applied to consecutive bits reducing it to the first case.

Lemma 5.3. *For any $k > 2$, and any m-bit matchcircuit Δ containing a k-bit matchgate Γ with character matrix A, there is a level $k-1$ matchcircuit Λ having the same character matrix as Δ.*

The proof for lemma 5.1-5.3 will be given in the full version.

5.1. Proof of theorem 3.3

Proof. For (1). Repeat the process in lemma 5.3 until there is no gate of bit greater than 2.

For (2). The number of matchgates used in the phases of transformation are $O(k)$, $O(k^3)$, $O(k)$ and $O(k^2)$, respectively, so a k-bit matchgate can be simulated by $O(k^4)$ many single and two-bit matchgates. This procedure is polynomial time computable, because there are polynomially many actions, and each action is polynomial time computable due to the fact that we compute only the edge entries. ∎

References

[1] J.-Y. Cai, V. Choudhary, On the Theory of Matchgate Computations. Electronic Colloquium on Computational Complexity (ECCC)(018), (2006).
[2] K. Murota, Matrices and Matroids for Systems Analysis, Springer, Berlin, 2000.
[3] Michael A. Nielsen and Isaac L. Chuang, Quantum Computation and Quantum Information, Combridge University Press, 2000.
[4] L. G. Valiant, Quantum circuits that can be simulated classically in polynomial time, SIAM Journal on Computing, **31** (2002) pp. 1229–1254.
[5] L. G. Valiant, Expressiveness of Matchgates, Theoretical Computer Science, **281** (2003) pp. 457–471.
[6] L. G. Valiant, Holographic Algorithms (Extended Abstract), FOCS 2004, pp. 306–315.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 503-514
www.stacs-conf.org

RENT, LEASE OR BUY: RANDOMIZED ALGORITHMS FOR MULTISLOPE SKI RENTAL

ZVI LOTKER [1], BOAZ PATT-SHAMIR [2], AND DROR RAWITZ [3]

[1] Dept. of Communication Systems Engineering, Ben Gurion University, Beer Sheva 84105 , Israel.

[2] School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel.

[3] Faculty of Science and Science Education & C.R.I., University of Haifa, Haifa 31905, Israel.
E-mail address: zvilo@cse.bgu.ac.il,boaz@eng.tau.ac.il,rawitz@cri.haifa.ac.il

Abstract. In the Multislope Ski Rental problem, the user needs a certain resource for some unknown period of time. To use the resource, the user must subscribe to one of several options, each of which consists of a one-time setup cost ("buying price"), and cost proportional to the duration of the usage ("rental rate"). The larger the price, the smaller the rent. The actual usage time is determined by an adversary, and the goal of an algorithm is to minimize the cost by choosing the best option at any point in time. Multislope Ski Rental is a natural generalization of the classical Ski Rental problem (where the only options are pure rent and pure buy), which is one of the fundamental problems of online computation. The Multislope Ski Rental problem is an abstraction of many problems where online decisions cannot be modeled by just two options, e.g., power management in systems which can be shut down in parts. In this paper we study randomized algorithms for Multislope Ski Rental. Our results include the best possible online randomized strategy for any *additive* instance, where the cost of switching from one option to another is the difference in their buying prices; and an algorithm that produces an e-competitive randomized strategy for any (non-additive) instance.

1. Introduction

Arguably, the "rent or buy" dilemma is the fundamental problem in online algorithms: intuitively, there is an ongoing game which may end at any moment, and the question is to commit or not to commit. Choosing to commit (the 'buy' option) implies paying large cost immediately, but low overall cost if the game lasts for a long time. Choosing not to commit (the 'rent' option) means high spending rate, but lower overall cost if the game ends quickly. This problem was first abstracted in the "Ski Rental" formulation [10] as follows. In the buy option, a one-time cost is incurred, and thereafter usage is free of charge. In the rent option, the cost is proportional to usage time, and there is no one-time cost. The deterministic solution is straightforward (with competitive factor 2). In the randomized

Key words and phrases: competitive analysis; ski rental; randomized algorithms.

The second author was supported in part by the Israel Science Foundation (grant 664/05) and by Israel Ministry of Science and Technology Foundation.

model, the algorithm chooses a random time to switch from the rent to the buy option (the adversary is assumed to know the algorithm but not the actual outcomes of random experiments). As is well known, the best possible online strategy for classical ski rental has competitive ratio of $\frac{e}{e-1} \approx 1.582$.

In many realistic cases, there may be some intermediate options between the extreme alternatives of pure buy and pure rent: in general, it may be possible to pay only a part of the buying cost and then pay only partial rent. The general problem, called here the *Multislope Ski Rental* problem, can be described as follows. There are several *states* (or *slopes*), where each state i is characterized by two numbers: a *buying cost* b_i and a *rental rate* r_i (see Fig. 1). Without loss of generality, we may assume that for all i, $b_i < b_{i+1}$ and $r_i > r_{i+1}$, namely that after ordering the states in increasing buying costs, the rental rates are decreasing. The basic semantics of the multislope problem is natural: to hold the resource under state i for t time units, the user is charged $b_i + r_i t$ cost units. An adversary gets to choose how long the game will last, and the task is to minimize total cost until the game is over.

The Multislope Ski Rental problem introduces entirely new difficulties when compared to the classical Ski Rental problem. Intuitively, whereas the only question in the classical version is when to buy, in the multislope version we need also to answer the question of what to buy. Another way to see the difficulty is that the number of potential transitions from one slope to another in a strategy is one less than the number of slopes, and finding a single point of transition is qualitatively easier than finding more than one such point.

In addition, the possibility of multiple transitions forces us to define the relation between multiple "buys." Following [2], we distinguish between two natural cases. In the *additive* case, buying costs are cumulative, namely to move from state i to state j we only need to pay the difference in buying prices $b_j - b_i$. In the *non-additive* case, there is an arbitrarily defined transition cost b_{ij} for each pair of states i and j.

Our results. In this paper we analyze randomized strategies for Multislope Ski Rental. (We use the term *strategy* to refer to the procedure that makes online decisions, and the term *algorithm* to refer to the procedure that computes strategies.) Our main focus is the additive case, and our main result is an efficient algorithm that computes the best possible randomized online strategy for any given instance of additive Multislope Ski Rental problem. We first give a simpler algorithm which decomposes a $(k+1)$-slope instance into k two-slope instances, whose competitive factor is $\frac{e}{e-1}$. For the non-additive model, we give a simple e-competitive randomized strategy.

Related Work. Variants of ski rental are implicit in many online problems. The classical (two-slope) ski rental problem, where the buying cost of the first slope and the rental rate of the second slope are 0, was introduced in [10], with optimal strategies achieving competitive factors of 2 (deterministic) and $\frac{e}{e-1}$ (randomized). Karlin et al. [9] apply the randomized strategy to TCP acknowledgment mechanism and other problems. The classical ski rental is sometimes called the *leasing* problem [5].

Azar et al. [3] consider a problem that can be viewed as non-additive multislope ski rental where slopes become available over time, and obtain an online strategy whose competitive ratio is $4 + 2\sqrt{2} \approx 6.83$. Bejerano et al. [4], motivated by rerouting in ATM networks, study the non-additive multislope problem. They give a deterministic 4-competitive strategy, and show that the factor of 4 holds assuming only that the slopes are concave, i.e., when the rent in a slope may decrease with time. Damaschke [6] considers a static version

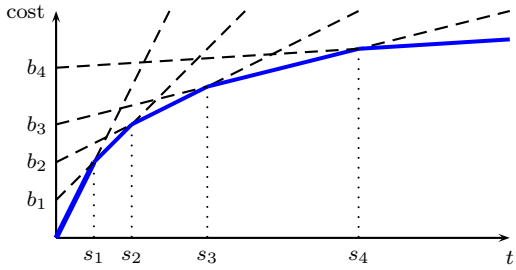


Figure 1: *A multislope ski rental instance with 5 slopes: The thick line indicates the optimal cost as a function of the game duration time.*

of the problem from [3], namely non-additive multislope ski rental problem where each slope is bought "from scratch."[1] For deterministic strategies, [6] gives an upper bound of 4 and a lower bound of $\frac{5+\sqrt{5}}{2} \approx 3.618$; [6] also presents a randomized strategy whose competitive factor is $2/\ln 2 = 2.88$. As far as we know, Damaschke's strategy is the only randomized strategy for multislope ski rental to appear in the literature.

Irani et al. [8] present a deterministic 2-competitive strategy for the additive model that generalizes the strategy for the two slopes case. They motivate their work by energy saving: each slope corresponds to some partial "sleep" mode of the system. Augustine et al. [2] present a dynamic program that computes the best deterministic strategy for non-additive multislope instances. The case where the length of the game is a stochastic variable with known distribution is also considered in both [8, 2].

Meyerson [12] defines the seemingly related "parking permit" problem, where there are k types of permits of different costs, such that each permit allows usage for some duration of time. Meyerson's results indicate that the problems are not very closely related, at least from the competitive analysis point of view: It is shown in [12] that the competitive ratio of the parking permit problem is $\Theta(k)$ and $\Theta(\log k)$ for deterministic and randomized strategies, respectively.

Organization. The remainder of this paper is organized as follows. In Section 2 we define the basic additive model and make a few preliminary observations. In Section 3 we give a simple algorithm to solve the multislope problem, and in Section 4 we present our main result: an optimal online algorithm. An e-competitive algorithm for the non-additive case is presented in Section 5.

2. Problem Statement and Preliminary Observations

In this section we formalize the *additive* version of the multislope ski rental problem. A k-ski rental instance is defined by a set of $k+1$ *states*, and for each state i there is a *buying cost* b_i and a *renting cost* r_i. A state can be represented by a line: the ith state corresponds to the line $y = b_i + r_i x$. Fig. 1 gives a geometrical interpretation of a multislope ski rental instance with five states. We use the terms "state" and "slope" interchangeably.

The requirement of the problem is to specify, for all times t, which slope is chosen at time t. We assume that state transitions can be only forward, and that states cannot be

[1] It can be shown that strategies that work for this case also work for the general non-additive case (see Section 5).

skipped, i.e., the only transitions allowed are of the type $i \to i+1$. We stress that this assumption holds without loss of generality in the additive model, where a transition from state $i \to j$ for $j > i+1$ is equivalent to a sequence of transitions $i \to i+1 \to \ldots \to j$ (cf. Section 5). It follows that a *deterministic strategy* for the additive multislope ski rental problem is a monotone non-decreasing sequence $(t_1, \ldots, t_k)$ where $t_i \in [0, \infty)$ corresponds to the transition $i-1 \to i$.

A *randomized strategy* can be described using a probability distribution over the family of deterministic strategies. However, in this paper we use another way to describe randomized strategies. We specify, for all times t, a probability distribution over the set of $k+1$ slopes. The intuition is that this distribution determines the actual cost paid by any online strategy. Formally, a *randomized profile* (or simply a *profile*) is specified by a vector $p(t) = (p_0(t), \ldots, p_k(t))$ of $k+1$ functions, where $p_i(t)$ is the probability to be in state i at time t. The correctness requirement of a profile is $\sum_{i=0}^{k} p_i(t) = 1$ for all $t \geq 0$. Clearly, any strategy is related to some profile. In the sequel we consider a specific type of profiles for which a randomized strategy can be easily obtained.

The performance of a profile is defined by its total accrued cost, which consists of two parts as follows. Given a randomized profile p, the expected *rental cost* of p at time t is

$$R_p(t) \stackrel{\text{def}}{=} \textstyle\sum_i p_i(t) \cdot r_i \ ,$$

and the expected total rental cost up to time t is

$$\int_{z=0}^{t} R_p(z)dz \ .$$

The second part of the cost is the buying cost. In this case it is easier to define the cumulative buying cost. Specifically, the expected *total buying cost* up to time t is

$$B_p(t) \stackrel{\text{def}}{=} \textstyle\sum_i p_i(t) \cdot b_i \ .$$

The expected *total cost* for p up to time t is

$$X_p(t) \stackrel{\text{def}}{=} B_p(t) + \int_{z=0}^{t} R_p(z)dz \ .$$

The goal of the algorithm is to minimize total cost up to time t for any given $t \geq 0$, with respect to the best possible. Intuitively, we think of a game that may end at any time. For any possible ending time, we compare the total cost of the algorithm with the best possible (offline) cost. To this end, consider the optimal solution of a given instance. If the games ends at time t, the optimal solution is to select the slope with the least cost at time t (the thick line in Fig. 1 denotes the optimal cost for any given t). More formally, the optimal offline cost at time t is

$$\textsc{opt}(t) = \min_i (b_i + r_i \cdot t) \ .$$

For $i > 0$, denote by s_i the time t instance where $b_{i-1} + r_{i-1} \cdot t = b_i + r_i \cdot t$, and define $s_0 = 0$. It follows that the optimal slope for a game ending at time t is the slope i for which $t \in [s_i, s_{i+1}]$ (if $t = s_i$ for some i then both slopes $i-1$ and i are optimal).

Finally, let us rule out a few trivial cases. First, note that if there are two slopes such that $b_i \leq b_j$ and $r_i \leq r_j$ then the cost incurred by slope j is never less than the cost incurred slope i, and we may therefore just ignore slope j from the instance. Consequently, we will assume henceforth, without loss of generality, that the states are ordered such that $r_{i-1} > r_i$ and $b_{i-1} < b_i$ for $1 \leq i \leq k$.

Second, using similar reasoning, note that we may consider only strategies that are monotone over time with respect to majorization [11], i.e., strategies such that for any two times $t \leq t'$ we have

$$\sum_{i=0}^{j} p_i(t) \geq \sum_{i=0}^{j} p_i(t') \ . \tag{2.1}$$

Intuitively, Eq. (2.1) means that there is no point is "rolling back" purchases: if at a given time we have a certain composition of the slopes, then at any later time the composition of slopes may only improve. Note that Eq. (2.1) implies that B_p is monotone increasing and R_p is monotone decreasing, i.e., over time, the strategy invests non-negative amounts in buying, resulting in decreased rental rates.

3. An $\frac{e}{e-1}$-Competitive Algorithm

In this section we describe how to solve the multislope problem by reducing it to the classical two-slope version, resulting in a randomized strategy whose competitive factor is $\frac{e}{e-1}$. This result serves as a warm-up and it also gives us a concrete upper bound on the competitiveness of the algorithm presented in Section 4.

The case of $r_k = 0$. Suppose we are given an instance (b, r) with $k+1$ slopes, where $r_k = 0$. We define the following k instances of the classical two-slopes ski rental problem: in instance i for $i \in \{1, \ldots, k\}$, we set

$$\text{instance } i\text{: } b_0^i = 0 \ \text{ and } \ r_0^i = r_{i-1} - r_i \ ; \ b_1^i = b_i - b_{i-1} \ \text{ and } \ r_1^i = 0 \ . \tag{3.1}$$

Observe that $b_1^i = r_0^i \cdot s_i$, i.e., the two slopes of the ith instance intersect exactly at s_i, their intersection point at the original multislope instance. Now, let $\textsc{opt}(t)$ denote the optimal offline solution to the original multislope instance, and let $\textsc{opt}^i(t)$ denote the optimal solution of the ith instance at time t, i.e., $\textsc{opt}^i(t) = \min\{b_1^i, r_0^i \cdot t\}$. We have the following.

Lemma 3.1. $\textsc{opt}(t) = \sum_{i=1}^{k} \textsc{opt}^i(t)$.

Proof. Consider a time t and let $i(t)$ be the optimal multislope state at time t. Then,

$$\begin{aligned} \sum_{i=1}^{k} \textsc{opt}^i(t) &= \sum_{i:s_i \leq t} b_1^i + \sum_{i:s_i > t} r_0^i \cdot t \\ &= \sum_{i:s_i \leq t} (b_i - b_{i-1}) + \sum_{i:s_i > t} (r_{i-1} - r_i) \cdot t \ = \ b_{i(t)} + r_{i(t)} \cdot t \ = \ \textsc{opt}(t) \ . \end{aligned}$$

■

Given the decomposition (3.1), it is easy to obtain a strategy for any multislope instance by combining strategies for k classical instances. Specifically, what we do is as follows. Let p^i be the $\frac{e}{e-1}$-competitive profile for the ith (two slope) instance (see [10]). We define a profile $\hat{p}$ for the multislope instance as follows: $\hat{p}_i(t) = p_1^i(t) - p_1^{i+1}(t)$ for $i \in \{1, \ldots, k-1\}$, $\hat{p}_0(t) = p_0^1(t)$, and $\hat{p}_k(t) = p_1^k(t)$. We first prove that the profile is well defined.

Lemma 3.2. *(1)* $p_1^i(t) \leq p_1^{i-1}(t)$ *for every* $i \in \{1, \ldots, k\}$ *and time* t. *(2)* $\sum_{i=0}^{k} \hat{p}_i(t) = 1$.

Proof. By the algorithm for classical ski rental, we have that the strategy for the i instance is $p_1^i(t) = (e^{t \cdot r_0^i / b_1^i} - 1)/(e-1)$. Claim (1) of the lemma now follows from that fact that $b_1^i / r_0^i = s_i > s_{i-1} = b_1^{i-1} / r_0^{i-1}$ for every $i \in \{1, \ldots, k\}$. Claim (2) follows from the telescopic sum

$$\sum_{i=0}^{k} \hat{p}_i(t) = p_0^1(t) + \sum_{i=1}^{k-1} (p_1^i(t) - p_1^{i+1}(t)) + p_1^k(t) = p_0^1(t) + p_1^1(t) = 1 \ .$$

■

Next, we show how to convert the profile $\hat{p}$ into a strategy. Note that the strategy uses a single random experiment, since arbitrary dependence between the different p_is are allowed.

Lemma 3.3. *Given $\hat{p}$ one can obtain an online strategy whose profile is $\hat{p}$.*

Proof. Define $\hat{P}_i(t) \stackrel{\text{def}}{=} \sum_{j \geq i} \hat{p}_j(t)$ and let U be a random variable that is chosen uniformly from $[0, 1]$. The strategy is as follows: we move from state i to state $i+1$ when $U = \hat{P}_i(t)$ for every state i. Namely, the ith transition time t_i is the time t such that $U = \hat{P}_i(t)$. ■

Thus we obtain the following:

Theorem 3.4. *The expected cost of the strategy defined by $\hat{p}$ is at most $\frac{e}{e-1}$ times the optimal offline cost.*

Proof. We first show that by linearity, the expected cost to the combined strategy is the sum of the costs to the two-slope strategies, i.e., that $X_{\hat{p}}(t) = \sum_{i=1}^{k} X_{p^i}(t)$. For example, the buying cost is

$$B_{\hat{p}}(t) = \sum_{i=0}^{k} \hat{p}_i(t) \cdot b_i = \sum_{i=0}^{k-1} (p_1^i(t) - p_1^{i+1}(t)) \cdot b_i + p_1^k(t) \cdot b_k = \sum_{i=1}^{k} p_1^i(t) \cdot (b_i - b_{i-1}) = \sum_{i=1}^{k} B_{p^i}(t) \ .$$

Similarly, $R_{\hat{p}}(t) = \sum_{i=1}^{k} R_{p^i}(t)$ by linearity, and therefore,

$$X_{\hat{p}}(t) = B_{\hat{p}}(t) + \int_{z=0}^{t} R_{\hat{p}}(z) dz = \sum_{i=1}^{k} B_{p^i}(t) + \int_{z=0}^{t} \left(\sum_{i=1}^{k} R_{p^i}(z) \right) dz = \sum_{i=1}^{k} X_{p^i}(t) \ .$$

Finally, by Lemma 3.1 and the fact that the strategies $p^1, \ldots, p^k$ are $\frac{e}{e-1}$-competitive we conclude that

$$X_{\hat{p}}(t) = \sum_{i=1}^{k} X_{p^i}(t) \leq \sum_{i=1}^{k} \frac{e}{e-1} \cdot \text{OPT}^i(t) = \frac{e}{e-1} \cdot \text{OPT}(t)$$

which means that $\hat{p}$ is $\frac{e}{e-1}$-competitive. ■

The case of $r_k > 0$. We note that if the smallest rental rate r_k is positive, then the competitive ratio is strictly less that $\frac{e}{e-1}$: this can be seen by considering a new instance where r_k is subtracted from all rental rates, i.e., $b'_i = b_i$ and $r'_i = r_i - r_k$ for all $0 \leq i \leq k$. Suppose p is $\frac{e}{e-1}$-competitive with respect to (r', k') (note that $r'_k = 0$). Then the competitive ratio of p at time t w.r.t. the original instance is:

$$c(t) = \frac{X_p(t)}{\text{OPT}(t)} = \frac{X'_p(t) + r_k \cdot t}{\text{OPT}'(t) + r_k \cdot t} \leq \frac{\frac{e}{e-1} \cdot \text{OPT}'(t) + r_k \cdot t}{\text{OPT}'(t) + r_k \cdot t} = \frac{e}{e-1} - \frac{1}{e-1} \cdot \frac{1}{\frac{\text{OPT}'(t)}{r_k \cdot t} + 1}$$

$\frac{d}{dt}\text{OPT}'(t) = r_i - r_k$ for $t \in [s_{i-1}, s_i)$. Hence, the ratio $\frac{\text{OPT}'(t)}{r_k \cdot t}$ is monotone decreasing, and thus $c(t)$ is monotone decreasing as well. It follows that

$$c \leq \frac{e}{e-1} - \frac{1}{e-1} \cdot \frac{1}{\frac{r_0 - r_k}{r_k} + 1} = \frac{e - r_k/r_0}{e-1}$$

Observe that $c = \frac{e}{e-1}$ when $r_k = 0$, and that $c = 1$ when $r_k = r_0$ (i.e., when $k = 0$).

4. An Optimal Online Algorithm

In this section we develop an optimal online strategy for any given additive multislope ski rental instance. We reduce the set of all possible strategies to a subset of much simpler strategies, which on one hand contains an optimal strategy, and on the other hand is easier to analyze, and in particular, allows us to effectively find such an optimal strategy.

Consider an arbitrary profile. (Recall that we assume w.l.o.g. that no slope is completely dominated by another.) As a first simplification, we confine ourselves to profiles where each p_i has only finitely many discontinuities. This allows us to avoid measure-theoretic pathologies without ruling out any reasonable solution within the Church-Turing computational model. It can be shown that we may consider only continuous profiles (details omitted).

So let such a profile $p = (p_0, \ldots, p_k)$ be given. We show that it can be transformed into a profile of a certain structure without increasing the competitive factor. Our chain of transformations is as follows. First, we show that it suffices to consider only simple profiles we call "prudent." Prudent strategies buy slopes in order, one by one, without skipping and without buying more than one slope at a time. We then define the concept of "tight" profiles, which are prudent profiles that spend money at a fixed rate relative to the optimal offline strategy. We prove that there exists a tight optimal profile. Furthermore, the best tight profile can be effectively computed: Given a constant c, we show how to check whether there exists a tight c-competitive strategy, and this way, using binary search on c, we can find the best tight strategy. Finally, we explain how to construct that profile and a corresponding strategy.

4.1. Prudent and Tight Profiles

Our main simplification step is to show that it is sufficient to consider only profiles that buy slopes consecutively one by one. Formally, *prudent* profiles are defined as follows.

Definition 4.1 (active slopes, prudent profiles)**.** A slope i is *active* at time t if $p_i(t) > 0$. A profile is called *prudent* if at all times there is either one or two consecutive active slopes.

At any given time t, at least one slope is active because $\sum_i p_i(t) = 1$ by the problem definition. Considering Eq. (2.1) as well, we see that a continuous prudent profile progresses from one slope to next without skipping any slope in between: once slope i is fully "paid for" (i.e., $p_i(t) = 1$), the algorithm will start buying slope $i + 1$.

We now prove that the set of continuous prudent profiles contains an optimal profile. Intuitively, the idea is that a non-prudent profile must have two non-consecutive slopes with positive probability at some time. In this case we can "shift" some probability toward a middle slope and only improve the overall cost.

Theorem 4.2. *If there exists a continuous c-competitive profile p for some $c \geq 1$, then there exists a prudent c-competitive profile $\tilde{p}$.*

Proof. Let $p = (p_0, \ldots, p_k)$ be a profile and suppose that all the p_is are continuous. It follows that B_p is also continuous. Define $\text{best}(t) = \max\{i : b_i \le B_p(t)\}$ and $\text{next}(t) = \min\{i : b_i \ge B_p(t)\}$. In words, $\text{best}(t)$ is the most expensive slope that is fully within the buying budget of p at time t, and $\text{next}(t)$ is the most expensive slope that is at least partially within the buying budget of p at time t. Obviously, $\text{best}(t) \le \text{next}(t) \le \text{best}(t) + 1$ for all t. Now, we define $\tilde{p}$ as follows:

$$\tilde{p}_i(t) = \begin{cases} \frac{b_{\text{next}} - B_p(t)}{b_{\text{next}} - b_{\text{best}}} & i = \text{best}(t) \text{ and } \text{best}(t) \neq \text{next}(t), \\ \frac{B_p(t) - b_{\text{best}}}{b_{\text{next}} - b_{\text{best}}} & i = \text{next}(t) \text{ and } \text{best}(t) \neq \text{next}(t), \\ 1 & i = \text{best}(t) = \text{next}(t), \\ 0 & \text{otherwise.} \end{cases}$$

It is not hard to verify that $\sum_i p_i(t) = 1$ for every time t. Furthermore, observe that $\tilde{p}$ is prudent, because B_p is continuous. It remains to show that $\tilde{p}$ is c-competitive. We do so by proving that $B_{\tilde{p}}(t) = B_p(t)$ and $R_{\tilde{p}}(t) \le R_p(t)$ for all t. First, directly from definitions we have

$$\begin{aligned} B_{\tilde{p}}(t) &= p_{\text{best}(t)}(t) \cdot b_{\text{best}(t)} + p_{\text{next}(t)}(t) \cdot b_{\text{next}(t)} \\ &= \frac{b_{\text{next}(t)} - B_p(t)}{b_{\text{next}(t)} - b_{\text{best}(t)}} \cdot b_{\text{best}(t)} + \frac{B_p(t) - b_{\text{best}(t)}}{b_{\text{next}(t)} - b_{\text{best}(t)}} \cdot b_{\text{next}(t)} \;=\; B_p(t) \; . \end{aligned}$$

Consider now rental payments. To prove that $R_{\tilde{p}}(t) \le R_p(t)$ for every time t we construct inductively a sequence of probability distributions $p = p^0, \ldots, p^\ell = \tilde{p}$. The first distribution p^0 is defined to be p. Suppose now that p^j is not prudent. Distribution p^{j+1} is obtained from p^j as follows. For any t such that there are two non-consecutive slopes with positive probability, let $i_1(t), i_2(t), i_3(t)$ be any three slopes such that $i_1(t) = \text{argmin}\{i : p_i^j(t) > 0\}$, $i_3(t) = \text{argmax}\{i : p_i^j(t) > 0\}$, and $i_1(t) < i_2(t) < i_3(t)$ (such $i_2(t)$ exists because p^j is not prudent). Define

$$p_i^{j+1}(t) = \begin{cases} p_i^j(t) - \frac{\Delta^j(t)}{b_{i_2(t)} - b_{i_1(t)}} & i = i_1(t), \\ p_i^j(t) + \frac{\Delta^j(t)}{b_{i_2(t)} - b_{i_1(t)}} + \frac{\Delta^j(t)}{b_{i_3(t)} - b_{i_2(t)}} & i = i_2(t), \\ p_i^j(t) - \frac{\Delta^j(t)}{b_{i_3(t)} - b_{i_2(t)}} & i = i_3(t), \\ p_i^j(t) & i \notin \{i_1(t), i_2(t), i_3(t)\} \end{cases}$$

where $\Delta^j(t) > 0$ is maximized so that $p_i^{j+1}(t) \ge 0$ for all i. Intuitively, we shift a maximal amount of probability mass from slopes $i_1(t)$ and $i_3(t)$ to the middle slope $i_2(t)$. The fact that $\Delta^j(t)$ is maximized means that we have either that $p_{i_1}^{j+1}(t) = 0$, or $p_{i_3}^{j+1}(t) = 0$, or both. In any case, we may already conclude that $\ell < k$. Also note that by construction, for all t we have $B_{p^{j+1}}(t) = \sum_i p_i^{j+1}(t) \cdot b_i = \sum_i p_i^j(t) \cdot b_i = B_{p^j}(t)$. Hence, $p^\ell = \tilde{p}$.

As to the rental cost, fix a time t, and consider now the rent paid by p^j and p^{j+1}:

$$\begin{aligned} &R_{p^j}(t) - R_{p^{j+1}}(t) \;= \\ &\quad= r_{i_1(t)} \frac{\Delta^j(t)}{b_{i_2(t)} - b_{i_1(t)}} - r_{i_2(t)} \left(\frac{\Delta^j(t)}{b_{i_2(t)} - b_{i_1(t)}} \frac{\Delta^j(t)}{b_{i_3(t)} - b_{i_2(t)}} \right) + r_{i_3(t)} \frac{\Delta^j(t)}{b_{i_3(t)} - b_{i_2(t)}} \\ &\quad= \Delta^j(t) \cdot \left(\frac{r_{i_1(t)} - r_{i_2(t)}}{b_{i_2(t)} - b_{i_1(t)}} - \frac{r_{i_2(t)} - r_{i_3(t)}}{b_{i_3(t)} - b_{i_2(t)}} \right) \;>\; 0 \end{aligned}$$

where the last inequality follows from the fact that if $i < j$, then $\frac{b_j - b_i}{r_i - r_j}$ is the x coordinate of the intersection point between the slopes i and j. ∎

Our next step is to consider profiles that invest in buying as much as possible under some spending rate cap. Our approach is motivated by the following intuitive observation.

Observation 4.3. Let p^1 and p^2 be two randomized prudent profiles. If $B_{p^1}(t) \geq B_{p^2}(t)$ for every t, then $R_{p^1}(t) \leq R_{p^2}(t)$ for every t.

In other words, investing available funds in buying as soon as possible results in lower rent, and therefore in more available funds. Hence, we define a class of profiles which spend money as soon as possible in buying, as long as there is a better slope to buy, namely as long as $p_k(t) < 1$.

Definition 4.4. Let $c \geq 1$. A prudent c-competitive profile p is called *tight* if $X_p(t) = c \cdot \text{OPT}(t)$ for all t with $p_k(t) < 1$.

Clearly, if the last slope is flat, i.e., $r_k = 0$, then it must be the case that $p_k(s_k) = 1$ for any profile with finite competitive factor: otherwise, the cost to the profile will grow without bound while the optimal cost remains constant. However, it is important to note that if $r_k > 0$, there may exist an optimal profile p that never buys the last slope, but still its expected spending rate as t tends to infinity is $c \cdot r_k$.

It is easy to see that a tight profile can achieve any achievable competitive factor.

Lemma 4.5. *If there exists a c-competitive prudent profile p for some $c \geq 1$, then there exists a c-competitive tight profile $\tilde{p}$.*

Proof. Let $\tilde{p}$ be the prudent profile satisfying $X_{\tilde{p}}(t) = c \cdot \text{OPT}(t)$ for all t for which $\tilde{p}_k(t) < 1$. We need to show that $\tilde{p}$ is feasible. Since by definition, p buys with any amount left, it suffices to show that for all t, the rent paid by p is at most $c \cdot \frac{d}{dt}\text{OPT}(t)$. Indeed, $R_{\tilde{p}}(t) \leq R_p(t)$ for every t due to Observation 4.3, and since p is c-competitive it follows that $R_p(t) \leq c \cdot \frac{d}{dt}\text{OPT}(t)$ and we are done. ∎

4.2. Constructing Optimal Online Strategies

We now use the results above to construct an algorithm that produces the best possible online strategy for the multislope problem. The idea is to guess a competitive factor c, and then try to construct a c-competitive tight profile. Given a way to test for success, we can apply binary search to find the optimal competitive ratio c to any desired precision.

The main questions are how to test whether a given c is feasible, and how to construct the profiles. We answer these questions together: given c, we construct a tight c-competitive profile until either we fail (because c was too small) or until we can guarantee success. In the remainder of this section we describe how to construct a tight profile p for a given competitive factor c.

We begin with analyzing the way a tight profile may spend money. Consider the situation at some time t such that $p_k(t) < 1$. Let j be the maximum index such that $s_j \leq t$. Then $\frac{d}{dt}\text{OPT}(t) = r_j$. Therefore, the spending rate of a tight profile at time t must be $c \cdot r_j$. If $j < k$, the tight profile may spend at rate $c \cdot r_j$ until time s_{j+1} (or until $p_k(t) = 1$), and if

$j = k$ the tight profile may continue spending at this rate forever. Hence, for $t \in (s_j, s_{j+1})$, we have

$$\frac{d}{dt}B_p(t) + R_p(t) = c \cdot \frac{d}{dt}\text{OPT}(t) = c \cdot r_j \ . \tag{4.1}$$

Since p is tight and therefore prudent, we also have, assuming $\text{best}(t) = i$ and $\text{next}(t) = i+1$, that

$$B_p(t) = p_i(t)b_i + p_{i+1}(t)b_{i+1} \ ,$$

and

$$R_p(t) = p_i(t)r_i + p_{i+1}(t)r_{i+1} \ .$$

Plugging the above equations into Eq. (4.1), we get

$$\frac{d}{dt}p_i(t)b_i + \frac{d}{dt}p_{i+1}(t)b_{i+1} + p_i(t)r_i + p_{i+1}(t)r_{i+1} = c \cdot r_j$$

Since p is prudent, $p_i(t) = 1 - p_{i+1}(t)$ and hence $\frac{d}{dt}p_i(t) = -\frac{d}{dt}p_{i+1}(t)$. It follows that

$$\frac{d}{dt}p_{i+1}(t) + p_{i+1}(t) \cdot \frac{r_{i+1} - r_i}{b_{i+1} - b_i} = \frac{c \cdot r_j - r_i}{b_{i+1} - b_i} \tag{4.2}$$

A solution to a differential equation of the form $y'(x) + \alpha y(x) = \beta$ where α and β are constants is $y = \frac{\beta}{\alpha} + \Gamma \cdot e^{-\alpha x}$, where Γ depends on the boundary condition. Hence in our case we conclude that

$$p_{i+1}(t) = \frac{c \cdot r_j - r_i}{r_{i+1} - r_i} + \Gamma \cdot e^{\frac{r_i - r_{i+1}}{b_{i+1} - b_i} \cdot t} \ , \tag{4.3}$$

and $p_i(t) = 1 - p_{i+1}(t)$, where the constant Γ is determined by the boundary condition.

Eq. (4.3) is our tool to construct p in a piecewise iterative fashion. For example, we start constructing p from $t = 0$ using $p_1(t) = \frac{c \cdot r_0 - r_0}{r_1 - r_0} + \Gamma \cdot e^{\frac{r_0 - r_1}{b_1 - b_0} \cdot t}$ and the boundary condition $p_1(0) = 0$. We get that $\Gamma = \frac{r_0(c-1)}{r_0 - r_1}$, i.e.,

$$p_1(t) = \frac{r_0(c-1)}{r_0 - r_1} \cdot (e^{\frac{r_0 - r_1}{b_1 - b_0} t} - 1) \ ,$$

and this holds for all $t \le \min(s_1, t_1)$, where t_1 is the solution to $p_1(t_1) = 1$.

In general, Eq. (4.2) remains true so long as there is no change in the spending rate and in the slope the profile p is buying. The spending rate changes when t crosses s_j, and the profile starts buying slope $i + 2$ when $p_{i+1}(t) = 1$.

We can now describe our algorithm. Given a ratio c, Algorithm FEASIBLE is able to construct the tight profile p or to determine that such a profile does not exist. It starts with the boundary condition $p_1(0) = 0$ and reveals the first part of the profile as shown above. Then, each time the spending rate changes or there is a change in $\text{best}(i)$ it moves to the next differential equation with a new boundary condition. After at most $2k$ such iterations it either computes a c-competitive tight profile p or discovers that such a profile is infeasible. Since we are able to test for success using Algorithm FEASIBLE, we can apply binary search to find the optimal competitive ratio to any desired precision.

We note that it is easy to construct a strategy that corresponds to any given prudent profile p, as described in the proof of Lemma 3.3. We conclude with the following theorem.

Theorem 4.6. *There exists an $O(k \log \frac{1}{\varepsilon})$ time algorithm that given an instance of the additive multislope ski rental problem for which the optimal randomized strategy has competitive ratio c, computes a $(c + \varepsilon)$-competitive strategy.*

Algorithm 1 – FEASIBLE$(c, \mathcal{M})$: true if the k-ski instance $\mathcal{M} = (b, r)$ admits competitive factor c

Let $s_i = \frac{b_i - b_{i-1}}{r_{i-1} - r_i}$ for each $1 \le i \le k$
Boundary_Condition $\leftarrow$ "$p_1(0) = 0$"
$j \leftarrow 0$; $i \leftarrow 1$
loop
  Define $p_i(t) = \frac{c \cdot r_j - r_{i-1}}{r_i - r_{i-1}} + \Gamma \cdot \exp(\frac{r_{i-1} - r_i}{b_i - b_{i-1}} \cdot t)$
  Try to solve for Γ using Boundary_Condition
  if no solution **then return** FALSE ▷ *possible escape if not feasible*
  $y \leftarrow p_i(s_j)$
  if $y < 1$ **then**
    Boundary_Condition $\leftarrow$ "$p_i(s_j) = y$"
    $j \leftarrow j + 1$ ▷ *continue at the next interval* $[s_j, s_{j_1}]$
  else
    Let x be such that $p_i(x) = 1$
    Boundary_Condition $\leftarrow$ "$p_{i+1}(x) = 0$"
    $i \leftarrow i + 1$ ▷ *move to next slope*
  end if
  if $i > k$ or $j \ge k$ **then return** TRUE ▷ *we're done*
end loop

5. An e-Competitive Strategy for the Non-Additive Case

In this section we consider the non-additive multislope ski rental problem. We present a simple randomized strategy which improves the best known competitive ratio from $2/\ln 2 = 2.88$ to e. Our technique is a simple application of randomized repeated doubling (see, e.g., [7]), used extensively in competitive analysis of online algorithms. For example, deterministic repeated doubling appears in [1], and a randomized version appears in [13].

Before presenting the strategy let us consider the details of the non-additive model. Augustine at el. [2] define a general non-additive model in which a transition cost b_{ij} is associated with every two states i and j, and show that one may assume w.l.o.g. that $b_{ij} = 0$ if $i > j$ and that $b_{ij} \le b_j$ for every $i < j$. Observe that we may further assume that $b_{ij} = b_j$ for every i and j, since the optimal (offline) strategy remains the unchanged. It follows that the strategies from [3, 4, 6] that were designed for the case of buying slopes "from scratch" also work for the general non-additive case.

We propose using the following iterative online strategy, which is similar to the one in [6], except for the choice of the "doubling factor." Specifically, the jth iteration is associated with a bound B_j on $\text{OPT}(\tau)$, where τ denotes the termination time of the game. We define $B_1 \stackrel{\text{def}}{=} \text{OPT}(s_1)/\alpha^X$, where $\alpha > 1$ and X is a chosen at random uniformly in $[0, 1)$. We also define $B_{j+1} = \alpha \cdot B_j$. Let $\tau_j = \text{OPT}^{-1}(B_j)$ and let i_j be the optimal offline state at time τ_j. In case there are two such states, i.e., $\tau_j = s_i$ for some i, we define $i_j = i - 1$. It follows that $i_1 = 0$. In the beginning of the jth iteration the online strategy buys i_j and stays in i_j until the this iteration ends. The jth iteration ends at time τ_j. Observe that the first iteration starts with $B_1 = \text{OPT}(s_1)$, namely we use slope 0 until s_1.

Theorem 5.1. *The expected cost of the strategy described above is at most e times the optimum.*

Proof. Observe that the first iteration starts with $B_1 = \text{OPT}(s_1)$, namely we use slope 0 until s_1, and hence, if the game ends during the first iteration, i.e., before s_1/α^X, then the online strategy is optimal. Consider now the case where the game ends at time $\tau \geq s_1/\alpha^X$, and suppose that $\tau \in [\tau_\ell, \tau_{\ell+1})$ for $\ell > 1$. In this case, the expected cost of the online strategy is bounded by

$$\begin{aligned}
\mathbf{E}\left[\sum_{j=1}^{\ell} \text{OPT}(\tau_j) + \text{OPT}(\tau)\right] &\leq \mathbf{E}\left[\sum_{j=1}^{\ell+1} \text{OPT}(\tau_j)\right] \leq \mathbf{E}\left[\frac{\alpha}{\alpha-1} \cdot \text{OPT}(\tau_{\ell+1})\right] \\
&= \mathbf{E}\left[\frac{\alpha^{2-X}}{\alpha-1} \cdot \text{OPT}(\tau)\right] \\
&= \frac{\alpha}{\alpha-1} \cdot \int_{x=0}^{1} \alpha^x dx \cdot \text{OPT}(\tau) = \frac{\alpha}{\ln \alpha} \cdot \text{OPT}(\tau)
\end{aligned}$$

By choosing $\alpha = e$ the competitive ratio is $\frac{\alpha}{\ln \alpha} = e$ as required. ∎

Acknowledgment

We thank Seffy Naor and Niv Buchbinder for stimulating discussions.

References

[1] J. Aspnes, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.

[2] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *45th IEEE Symp. on Foundations of Computer Science*, pages 530–539, 2004.

[3] Y. Azar, Y. Bartal, E. Feuerstein, A. Fiat, S. Leonardi, and A. Rosén. On capital investment. *Algorithmica*, 25(1):22–36, 1999.

[4] Y. Bejerano, I. Cidon, and J. S. Naor. Dynamic session management for static and mobile users: a competitive on-line algorithmic approach. In *4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 65–74. ACM, 2000.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[6] P. Damaschke. Nearly optimal strategies for special cases of on-line capital investment. *Theoretical Computer Science*, 302(1-3):35–44, 2003.

[7] S. Gal. *Search Games*. Academic Press, 1980.

[8] S. Irani, R. K. Gupta, and S. K. Shukla. Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 117–123, 2002.

[9] A. R. Karlin, C. Kenyon, and D. Randall. Dynamic TCP acknowledgment and other stories about $e/(e-1)$. *Algorithmica*, 36(3):209–224, 2003.

[10] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):77–119, 1988.

[11] A. W. Marshall and I. Olkin. *Inequalities: Theory of Majorization and Its Applications*. Academic Press, 1979.

[12] A. Meyerson. The parking permit problem. In *46th IEEE Symp. on Foundations of Computer Science*, pages 274–284, 2005.

[13] R. Motwani, S. Phillips, and E. Torng. Non-clairvoyant scheduling. *Theor. Comput. Sci.*, 130(1):17–47, 1994.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 515-526
www.stacs-conf.org

LOWER BOUNDS FOR ADAPTIVE LINEARITY TESTS

SHACHAR LOVETT

Faculty of Mathematics and Computer Science
The Weizmann Institute of Science, POB 26
Rehovot 76100, Israel
E-mail address: Shachar.Lovett@weizmann.ac.il

Abstract. Linearity tests are randomized algorithms which have oracle access to the truth table of some function f, and are supposed to distinguish between linear functions and functions which are far from linear. Linearity tests were first introduced by Blum, Luby and Rubenfeld in [BLR93], and were later used in the PCP theorem, among other applications. The quality of a linearity test is described by its correctness c - the probability it accepts linear functions, its soundness s - the probability it accepts functions far from linear, and its query complexity q - the number of queries it makes.

Linearity tests were studied in order to decrease the soundness of linearity tests, while keeping the query complexity small (for one reason, to improve PCP constructions). Samorodnitsky and Trevisan constructed in [ST00] the Complete Graph Test, and prove that no Hyper Graph Test can perform better than the Complete Graph Test. Later in [ST06] they prove, among other results, that no non-adaptive linearity test can perform better than the Complete Graph Test. Their proof uses the algebraic machinery of the Gowers Norm. A result by Ben-Sasson, Harsha and Raskhodnikova [BHR05] allows to generalize this lower bound also to adaptive linearity tests.

We also prove the same optimal lower bound for adaptive linearity test, but our proof technique is arguably simpler and more direct than the one used in [ST06]. We also study, like [ST06], the behavior of linearity tests on quadratic functions. However, instead of analyzing the Gowers Norm of certain functions, we provide a more direct combinatorial proof, studying the behavior of linearity tests on random quadratic functions. This proof technique also lets us prove directly the lower bound also for adaptive linearity tests.

1. Introduction

We study the relation between the number of queries and soundness of adaptive linearity tests. A linearity test (over the field $\mathbb{F}_2$ for example) is a randomized algorithm which has oracle access to the truth table of a function $f : \{0,1\}^n \to \{0,1\}$, and needs to distinguish between the following two extreme cases:

(1) f is linear
(2) f is far from linear functions

1998 ACM Subject Classification: F.2.2, G.2.1.
Key words and phrases: Property testing, Linearity testing, Adaptive tests, Lower bounds.
Research supported by grant 1300/05 from the Israel Science Foundation.

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

A function f is called *linear* if it can be written as $f(x_1, ..., x_n) = a_1x_1 + ... + a_nx_n$, with $a_1, ..., a_n \in \mathbb{F}_2$. The agreement of two functions $f, g : \{0,1\}^n \to \{0,1\}$ is defined as $d(f,g) = |\mathbb{P}_{\mathbf{x}}[f(\mathbf{x}) = g(\mathbf{x})] - \mathbb{P}_{\mathbf{x}}[f(\mathbf{x}) \neq g(\mathbf{x})]|$. f is far from linear functions if it has small agreement with all linear functions (we make this definition precise in Section 2).

Linearity tests were first introduced by Blum, Luby and Rubenfeld in [BLR93]. They presented the following test (coined the BLR test), which makes only 3 queries to f:

(1) Choose $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ at random
(2) Verify that $f(\mathbf{x} + \mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$.

Bellare et al. [BCH+96] gave a tight analysis of the BLR test. It is obvious that the BLR test always accepts a linear function. They have shown that if the test accepts a function f with probability $1/2 + \epsilon$, then f has agreement at least 2ϵ with some linear function.

For a linearity test, we define that it has *completeness* c if it accepts any linear function with probability of at least c. A test has *perfect completeness* if $c = 1$. A linearity test has *soundness* s if it accepts any function f with agreement at most ϵ with all linear functions, with probability of at most $s + \epsilon'$, where $\epsilon' \to 0$ when $\epsilon \to 0$. We define the *query complexity* q of a test as the maximal number of queries it performs. In the case of the BLR test, it has perfect completeness, soundness $s = 1/2$ (with $\epsilon' = 2\epsilon$) and query complexity $q = 3$.

If one repeats a linearity test with query complexity q and soundness s independently t times, the query complexity grows to $q' = qt$ while the soundness reduces to $s' = s^t$. So, it makes sense to define the *amortized query complexity* $\bar{q}$ of a test as $\bar{q} = q/\log_2(1/s)$. Independent repetition of a test doesn't change it's amortized query complexity. Notice that the BLR test has amortized query complexity $\bar{q} = 3$.

Linearity tests are a key ingredient in the PCP theorem, started in the works of Arora and Safra [AS98] and Arora, Lund, Motwani, Sudan and Szegedy [ALM+98]. In order to improve PCP constructions, linearity tests were studied in order to improve their amortized query complexity.

Samorodnitsky and Trevisan [ST00] have generalized the basic BLR linearity test. They introduced the *Complete Graph Test*. The Complete Graph Test (on k vertices) is:

(1) Choose $\mathbf{x}_1, ..., \mathbf{x}_k \in \{0,1\}^n$ independently
(2) Verify $f(\mathbf{x}_i + \mathbf{x}_j) = f(\mathbf{x}_i) + f(\mathbf{x}_j)$ for all i, j

This test has perfect completeness and query complexity $q = \binom{k}{2} + k$. They show that all the $\binom{k}{2}$ tests that the Complete Graph Test performs are essentially independent, i.e. that the test has soundness $s = 2^{-\binom{k}{2}}$. This makes this test have amortized query complexity $\bar{q} = 1 + \theta(1/\sqrt{q})$. They show that this test is optimal among the family of Hyper-Graph Tests (see [ST00] for definition of this family of linearity tests), and raise the question of whether the Complete Graph Test is optimal among all linearity tests, i.e. does a test with the same query complexity but with better soundness exist?

They partially answer this question in [ST06], where (among many other results) they show that no non-adaptive linearity test can perform better than the Complete Graph Test. A test is called *non-adaptive* if it first chooses q locations in the truth table of f, then queries them, and based on the results accept or rejects f. Otherwise, a test is called *adaptive*. An adaptive test may decide on its query locations based on the values of f in previous queries.

The proof technique of [ST06] uses the algebraic analysis of the Gowers Norm of certain functions. The Gowers Norm is a measure of local closeness of a function to a low degree

polynomial. For more details regarding the definition and properties of the Gowers Norm, see [GT05] and [Sam07].

Ben-Sasson, Harsha and Raskhodnikova prove in [BHR05] that any adaptive linearity test with completeness c, soundness s and query complexity q can be transformed into a non-adaptive linearity test with the same query complexity, perfect completeness and soundness $s' = s + 1 - c$. Combining their result with the result of [ST06] proves the lower bound also for adaptive linearity tests.

We also prove the same optimal lower bound for adaptive linearity test, but our proof technique is arguably simpler and more direct than the one used in [ST06]. We also study, like [ST06], the behavior of linearity tests on quadratic functions. However, instead of employing algebraic analysis of the Gowers Norm of certain functions, we provide a more direct combinatorial proof, studying the behavior of linearity tests on random quadratic functions. This proof technique also lets us prove directly the lower bound also for adaptive linearity tests.

1.1. Our techniques

We model adaptive tests using test trees. A test tree T is a binary tree, where in each inner vertex v there is some label $\mathbf{x}(v) \in \{0,1\}^n$, and the leaves are labeled with either *accept* or *reject*. Running a test tree on a function f is done by querying at each stage f on the label of the current vertex (starting at the root), and following one of the two edges leaving the vertex, depending on the query response. When reaching a leaf, its label (*accept* or *reject*) is the value of that f gets in T . An adaptive test $\mathbb{T}$ can always be modeled as first randomly choosing a test tree from some set $\{T_i\}$, according to some distribution on the test trees, then running the test tree on f.

It turns out that in order to prove a lower bound which matches the upper bound of the Complete Graph Test, it is enough to consider functions f which are quadratic. Actually, it's enough to consider f which is a random quadratic function.

A function f is quadratic if it can be presented as $f(x_1, ..., x_n) = \sum_{i,j} a_{i,j} x_i x_j + \sum_i b_i x_i + c$ for some values $a_{i,j}, b_i, c \in \mathbb{F}_2$. We study the behavior of running test trees on a random linear function, and on a random quadratic function.

The main idea is as follows. Let v be some inner vertex in a test tree T, with the path from the root of T to v being $v_0, ..., v_{k-1}, v$. If $\mathbf{x}(v)$ is linearly dependent on $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$, then when running T on any linear function, the value of $f(\mathbf{x}(v))$ can be deduced from the already known values of $f(\mathbf{x}(v_0)), ..., f(\mathbf{x}(v_{k-1}))$. Therefore, if the vertex v is reached, then the same edge leaving v will always be taken by any linear function. Additionally, if $\mathbf{x}(v)$ is linearly independent of $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$, then either v is never reached running T on linear functions, or the two edges leaving v are taken with equal probability when running T on a random linear function. A similar analysis can be made when running T on quadratic functions, replacing *linear dependence* with a corresponding notion of *quadratic dependence*.

Using this observation, we can define the *linear rank* of a leaf v, marked $l(v)$, as the linear rank of labels on the path from the root to v. We prove that running the test tree T on a random linear function reaches v with probability $2^{-l(v)}$. Similarly, we define the *quadratic rank* of a leaf v, marked $q(v)$, as the quadratic rank of those labels, and we proving that running T on a random quadratic function reaches v with probability $2^{-q(v)}$. We prove that the quadratic rank of any set cannot be much larger than its linear rank,

and in particular that $q(v) \leq \binom{l(v)}{2} + l(v)$ for all leaves v. We use this inequality to prove that a test which has completeness c and query complexity q accepts a random quadratic function with a probability of at least $c - 1 + 2^{-q+\phi(q)}$, where $\phi(q)$ is defined as the unique non-negative solution to $\binom{\phi(q)}{2} + \phi(q) = q$.

We use this to show that any linearity test with completeness c and query complexity q must have $s \geq 2^{-q+\phi(q)}$. In particular, the Complete Graph Test on k vertices has perfect completeness, soundness $s = 2^{-\binom{k}{2}}$ and query complexity $q = \binom{k}{2} + k$. Since $\phi(q) = k$ the Complete Graph Test is optimal among all adaptive tests with the same query complexity.

In fact, we prove a stronger claim. We say that a test $\mathbb{T}$ has *average query complexity* q if for any function f, the average number of queries performed is at most q. In particular any test with query complexity q also has average query complexity q. We prove that for any test with completeness c and average query complexity q, the soundness is at least $s \geq 2^{-q+\phi(q)}$.

We present and analyze linearity tests over $\mathbb{F}_2$. Linearity tests can also be considered over larger fields or groups. Our lower bound actually generalizes easily to any finite field, but for ease of presentation, and since the techniques are exactly the same, we present everything over $\mathbb{F}_2$. We comment further on the modifications required for general finite fields in Section 2.

2. Preliminaries

2.1. Linearity tests

We call a function $f : \{0,1\}^n \to \{0,1\}$ linear if it can be written as $f(x_1, ..., x_n) = a_1x_1 + ... + a_nx_n$ for some $a_1, ..., a_n \in \{0,1\}$ where addition and multiplication are in $\mathbb{F}_2$.

A linearity test is a randomized algorithm with oracle access to the truth table of f, which is supposed to distinguish the following two extreme cases:

(1) f is linear (accept)
(2) f is ϵ-far from linear functions (reject)

where the agreement of two functions $f, g : \{0,1\} \to \{0,1\}$ is defined as $d(f,g) = |Pr_{\mathbf{x}}[f(\mathbf{x}) = g(\mathbf{x})] - Pr_{\mathbf{x}}[f(\mathbf{x}) \neq g(\mathbf{x})]|$, and f is ϵ-far from linear functions if the agreement it has with any linear function is at most ϵ.

We now follow with some standard definition regarding linearity tests (or more generally, property tests). We say a test has *completeness* c if for any linear function f the test accepts with probability at least c. A test has *perfect completeness* if $c = 1$. We say a test has *soundness* s if for any f which is ϵ-far from linear the test accepts with probability at most $s + \epsilon'$, where $\epsilon' \to 0$ when $\epsilon \to 0$ (in fact, we talk about a family of linearity tests, for $n \to \infty$, but we ignore this subtle point).

A test is said to have *query complexity* q if it accesses the truth-table of f at most q times (for any choice of it's internal randomness). A test is said to have *average query complexity* q if for any function f, the average number of accesses (over the internal randomness of the test) done to the truth table of f is at most q. Obviously, any test with query complexity q is also a test with average query complexity q.

We say a test is *non-adaptive* if it chooses all the locations it's going to query in the truth table of f before reading any of their values. Otherwise, we call the test *adaptive*.

We now turn to model adaptive tests in a way that will be more convenient for our analysis. We first define a test tree and running a test tree on a function.

Definition 2.1. A *test tree* on functions $\{0,1\}^n \to \{0,1\}$ is a rooted binary tree T. On each inner vertex of the tree v there is a label $x(v) \in \{0,1\}^n$. On each leaf there is a label of either *accept* or *reject*.

Definition 2.2. *Running a test tree T on a function f* is done as follows. We start at the root of the tree v_0, read the value of $f(x(v_0))$, and according to the value take the left or the right edge leaving v_0. We continue in this fashion on inner vertices of T until we reach a leaf of T. The *value of f in T* is the value of the end leaf (i.e. *accept* or *reject*), and the *depth of f in T* is the depth of the end vertex of f in T.

Using these definitions, we can now model adaptive tests. We identify an adaptive test $\mathbb{T}$ on functions $\{0,1\}^n \to \{0,1\}$ with a distribution of binary trees $\{T_i\}$ (also on functions $\{0,1\}^n \to \{0,1\}$). Running the test $\mathbb{T}$ on a function f is done by randomly choosing one of the trees T_i (according to their distribution), and then running the test tree T_i on f. The result of the function f in the test tree T_i is the result the test $\mathbb{T}$ returns on f.

Notice that a test has query complexity q iff all trees T_i has depth at most q, and has average query complexity q iff for any function f, the average depth reached in a random tree from $\{T_i\}$ is at most q.

In order to define our main theorem, we will define the following function. For $x > 0$ define $\phi(x)$ as the unique real positive solution to $\phi(x)^2/2 + \phi(x)/2 = x$. Notice that for positive integer $\phi(x)$, this is the same as $\binom{\phi(x)}{2} + \phi(x) = x$. The following is the main theorem of this paper:

Theorem 2.3. *(main theorem) Let $\mathbb{T}$ be an adaptive test with completeness c, soundness s and average query complexity $q \geq 1$. Then $s + 1 - c \geq 2^{-q+\phi(q)}$.*

Notice that for large q, $\phi(q) \approx \sqrt{2q}$, also $\sqrt{q} \leq \phi(q) \leq \sqrt{2q}$, so we get that in particular, $s + 1 - c \geq 2^{-q+\theta(\sqrt{q})}$.

The Complete Graph Test was presented in [ST00]. The test (on a graph with k vertices) can be described as choosing $\mathbf{x}_1, ..., \mathbf{x}_k$ at random, and querying f at $\mathbf{x}_i$ (for $i = 1..k$) and on $\mathbf{x}_i + \mathbf{x}_j$ (for $1 \leq i < j \leq k$). The test accepts f if for any i, j

$$f(x_i) + f(x_j) + f(x_i + x_j) = 0$$

In [ST00] it is proven that the Complete Graph Test has perfect completeness and soundness $s = 2^{-\binom{k}{2}}$. The total number of queries performed is $q = k + \binom{k}{2}$, so by our definitions, $k = \phi(q)$ and $s = 2^{-q+\phi(q)}$. We have the following corollary:

Corollary 2.4. *The Complete Graph Test is optimal among all adaptive linearity tests.*

Remark 2.5. We state and prove all results for functions $f : \{0,1\}^n \to \{0,1\}$. In fact, the lower bound result on adaptive linearity tests holds for functions $f : \mathbb{F}^n \to \mathbb{F}$ for any finite field $\mathbb{F}$, and not just $\mathbb{F}_2$, with only minor adjustments to the definitions and proofs. We need to make the following modifications:

(1) Define "ϵ-far from linear functions" for general fields
(2) Test trees should have $|F|$ edges leaving each edge instead of 2
(3) The proof that random quadratic functions are far from linear, proved in Section 5, should be slightly modified

Since the results follow simply for any finite field, we chose to present the results over $\mathbb{F}_2$ to make the presentation simpler and clearer.

3. Quadratic functions

We will see that in order to prove Theorem 2.3, it will be enough to limit the functions f to be quadratic. We say a function f is quadratic if it can be written as:

$$f(x_1, ..., x_n) = \sum_{i,j} a_{i,j} x_i x_j + \sum_i b_i x_i + c$$

for some $a_{i,j}, b_i, c \in \mathbb{F}_2$.

In fact, for our usage, we will force our quadratic functions f to have $f(0) = 0$ (equivalently, $c = 0$ in the above description). So, throughout this paper, when speaking of quadratic functions, we actually speak of quadratic functions f with the added condition $f(0) = 0$.

We will study the dynamics of a test tree T in a linearity test $\mathbb{T}$, in two cases - when applied to a uniformly random linear function, and when applied to a uniformly random quadratic function.

The following technical lemma is the key ingredient to the proof of the Theorem 2.3.

Lemma 3.1. *Let $\mathbb{T}$ be an adaptive linearity test with completeness c and average query complexity q. Then running $\mathbb{T}$ on a random quadratic function returns* accept *with probability at least $c - 1 + 2^{-q+\phi(q)}$.*

In order to prove Theorem 2.3, we will also need the following simple lemma:

Lemma 3.2. *Let f be a random quadratic function. Then the probability that f is not $2^{-\Omega(n)}$-far from linear functions is $2^{-\Omega(n)}$.*

Theorem 2.3 now follows directly from Lemmas 3.1 and 3.2. We sketch now it's proof following the two lemmas.

Proof. (of the main theorem) The average probability that $\mathbb{T}$ returns *accept* on a random quadratic function which is $2^{-\Omega(n)}$-far from linear functions is at least $c-1+2^{-q+\phi(q)}-2^{-\Omega(n)}$. So, there exists some quadratic function f which is $2^{-\Omega(n)}$-far from linear and on which $\mathbb{T}$ returns *accept* with probability at least $c - 1 + 2^{-q+\phi(q)} - 2^{-\Omega(n)}$. Taking $n \to \infty$ shows that $s + 1 - c \geq 2^{-1+\phi(q)}$. ∎

The remainder of the paper is organized as follows. Lemma 3.1 is proved in Section 4, and Lemma 3.2 is proved in Section 5.

4. Linearity test applied to a random quadratic function

We study tests and test trees applied to linear and quadratic functions, in order to prove Lemma 3.1. Let $\mathbb{T}$ be an adaptive test with completeness c and average query complexity q. Let T be a some test tree which is a part of the test $\mathbb{T}$.

We start by studying the dynamics of applying T to linear functions. Assume we know that f is a linear function, and we are at some vertex $v \in T$, where the path from the root to v is $v_0, .., v_{k-1}, v$. Assume $\mathbf{x}(v)$ is linearly dependant on $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$. Since we know f is linear, we can deduce the value of $\mathbf{x}(v)$ from $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$, and so we will always

follow the same edge leaving v when we apply T to any linear function. On the other hand, if $\mathbf{x}(v)$ is linearly independent of $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$, we know that when we apply T to a random linear function, either we never reach v, or we have equal chances of taking any of the two edges leaving v.

This gives rise to the following formal definition:

Definition 4.1. Let v be a leaf in T, where the path from the root to v is $v_0, v_1, ..., v_{k-1}, v$. We define the *linear degree* of v, marked $l(v)$, to be the linear rank of $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$.

We define L_T to be the set of leaves of T to which linear functions can arrive. i.e, $v \in L$ if the path from the root to v, $v_0, ..., v_{k-1}, v$ always takes the "correct" edge leaving any vertex v_i with $\mathbf{x}(v_i)$ linearly dependent on $\mathbf{x}(v_0), ..., \mathbf{x}(v_{i-1})$.

The following lemma formalizes the discussion above:

Lemma 4.2. *For any test tree T:*

(1) *For any $v \in L_T$, the probability that a random linear function will arrive to v is $2^{-l(v)}$*

(2) $\sum_{v \in L_T} 2^{-l(v)} = 1$

For $v \in L_T$, we define $c(v)$ to be 1 if the value of v is *accept*, and $c(v) = 0$ otherwise. Since the completeness of $\mathbb{T}$ is c, we have that the probability that $\mathbb{T}$ returns *accept* on a random linear function is at least c. On the other hand, for any test tree T in $\mathbb{T}$, the probability that a random linear function will return *accept* is exactly $\sum_{v \in L_T} c(v) 2^{-l(v)}$. So, the following lemma follows:

Lemma 4.3. $\mathbb{E}_T \sum_{v \in L_T} c(v) 2^{-l(v)} \geq c$

where by $\mathbb{E}_T$ here and throughout the paper we mean the average value of a random test tree T in $\mathbb{T}$.

We now generalize the concept of linear dependence to quadratic functions.

Definition 4.4. Let $\mathbf{x}_1, ..., \mathbf{x}_k \in \{0,1\}^n$.

(1) We say $\mathbf{x}_1, ..., x_k$ are *quadratically dependent* if there are constants $a_1, ..., a_k \in \mathbb{F}_2$, not all zero, s.t. for any quadratic function f we have: $a_1 f(\mathbf{x}_1) + ... + a_k f(\mathbf{x}_k) = 0$. otherwise will call $x_1, ..., x_k$ *quadratically independent*.

(2) We say $\mathbf{x}_k$ is *quadratically dependent* on $\mathbf{x}_1, ..., \mathbf{x}_{k-1}$ if there are constants $a_1, ..., a_{k-1} \in \mathbb{F}_2$ s.t. for any quadratic function f we have: $f(\mathbf{x}_k) = a_1 f(\mathbf{x}_1) + ... + a_{k-1} f(\mathbf{x}_{k-1})$. Otherwise we say $\mathbf{x}_k$ is *quadratically independent* of $\mathbf{x}_1, ..., \mathbf{x}_{k-1}$.

(3) We define the *quadratic dimension* of $\mathbf{x}_1, ..., \mathbf{x}_k$ to be the size of the largest subset of $\{\mathbf{x}_1, ..., \mathbf{x}_k\}$ which is quadratically independent.

This definition may seem obfuscated, but the following alternative yet equivalent definition will clarify it. The space of quadratic functions over $\{0,1\}^n$ is a linear space over $\mathbb{F}_2$. Let M be it's generating matrix, i.e. the rows of M are a base for the linear space (in particular, the dimensions of M are $(\binom{n}{2} + n) \times 2^n$). A column of M corresponds to an input $\mathbf{x} \in \{0,1\}^n$. Now, $\mathbf{x}_1, ..., \mathbf{x}_k$ are quadratically dependent iff the columns corresponding to them are linearly dependent, and similarly for the other definitions.

Notice that the usual definition of linear dependence is equivalent to this more complex definition, when applied to the linear space of all linear functions.

We now can repeat the informal discussion at the start of this section, except this time for quadratic functions, with all the reasoning left intact. Let $v \in T$ be a vertex, with path from the root being $v_0, ..., v_{k-1}, v$. Assume $\mathbf{x}(v)$ is quadratically dependent on $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$, and f is any quadratic function. The value of $f(\mathbf{x}(v))$ can be deduced from the already known values of $f(\mathbf{x}(v_0)), ..., f(\mathbf{x}(v_{k-1}))$, and so only one edge leaving v will be taken on all quadratic functions. Alternatively, if $x(v)$ is quadratically independent on $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$, then a random quadratic function either never reaches v, or has equal chances of taking each of the two edges leaving v.

This leads to the following definition and lemma for quadratic degree of a vertex $v \in T$, similar to the ones for linear degree.

Definition 4.5. Let v be a leaf in T, where the path from the root to v is $v_0, v_1, ..., v_{k-1}, v$. We define the *quadratic degree* of v, marked $q(v)$, to be the quadratic rank of $\mathbf{x}(v_0), ..., \mathbf{x}(v_{k-1})$.

We define Q_T to be the set of leaves of T to which quadratic functions can arrive. Naturally $L_T \subseteq Q_T$. The following lemma on quadratic degree follows from the discussion above:

Lemma 4.6. *For any test tree T:*

(1) *For any $v \in Q_T$, the probability that a random quadratic function will arrive to v is $2^{-q(v)}$*
(2) $\sum_{v \in Q} 2^{-q(v)} = 1$
(3) *For any $v \in L_T$ we have $q(v) \geq l(v)$*

Last, we mark the depth of a vertex $v \in T$ by $d(v)$. Since $\mathbb{T}$ has average query complexity q, we know that for any function f, the average depth of running a random tree T of $\mathbb{T}$ on f is at most q. So, this also holds for a random linear function. However, the average depth a random linear function arrives on a tree T is exactly $\sum d(v)2^{-l(v)}$, so the following lemma follows.

Lemma 4.7. $\mathbb{E}_T \sum_{v \in L_T} d(v)2^{-l(v)} \leq q$

We now wish to make a connection between $q(v)$ and $l(v)$ for vertices $v \in L_T$.

First, we prove that following lemma:

Lemma 4.8. *For any $\boldsymbol{x}_1, ..., \boldsymbol{x}_k \in \{0,1\}^n$ there are coefficients $a_{i,j}, b_i \in \mathbb{F}_2$ s.t. for any quadratic function f we have:*

$$f(\boldsymbol{x}_1 + ... + \boldsymbol{x}_k) = \sum_{i,j} a_{i,j} f(\boldsymbol{x}_i + \boldsymbol{x}_j) + \sum_i b_i f(\boldsymbol{x}_i)$$

Proof. Let $f(\mathbf{x})$ by some polynomial of degree d. It's derivative in the $\mathbf{y}$ direction is defined to be $f_{\mathbf{y}}(\mathbf{x}) = f(\mathbf{x}+\mathbf{y}) - f(\mathbf{x})$. It's easy to see that the degree of $f_{\mathbf{y}}$ as a function of $\mathbf{x}$ is at most $d-1$. So, taking 3 derivatives from a quadratic function makes it the zero function, and so in particular for any quadratic function f, we we take it's derivatives in directions $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$, and evaluate the result at 0, we get that

$$(((f_{\mathbf{x}})_{\mathbf{y}})_{\mathbf{z}}(0) = 0$$

Opening this expression yields:

$$f(\mathbf{x}+\mathbf{y}+\mathbf{z}) - f(\mathbf{x}+\mathbf{y}) - f(\mathbf{x}+\mathbf{z}) - f(\mathbf{y}+\mathbf{z}) + f(\mathbf{x}) + f(\mathbf{y}) + f(\mathbf{z}) - f(0) = 0$$

Since $f(0) = 0$, we can express $f(\mathbf{x}+\mathbf{y}+\mathbf{z})$ as a sum of application of f on an element, or sum of two elements in $\{x, \mathbf{y}, \mathbf{z}\}$. This proves the lemma for $k = 3$. For $k > 3$ we use simple induction. ∎

Now we can bound $l(v)$ in term of $q(v)$. We first prove a result bounding in general the linear rank of a set by it's quadratic rank.

Lemma 4.9. *Let $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_k\}$ be elements in $\{0,1\}^n$. Let l be the their linear rank, and q their quadratic rank. Then*

$$q \leq \binom{l}{2} + l$$

Proof. Let $S \subset \{\mathbf{x}_1, ..., \mathbf{x}_k\}$ be a maximal quadratic independent set. $|S| = q$. The linear rank of S is also l. Let $S' \subset S$ be a maximal set of linearly independent elements of S. $|S'| = l$. Assume w.l.o.g that $S' = \{\mathbf{x}_1, ..., \mathbf{x}_l\}$. Since every $x \in S$ is linearly dependent on S', it can be written as a sum of some of the elements of S'. Using Lemma 4.8, we get that for any $\mathbf{x} \in S$ there exists coefficients $a_{i,j}^{(\mathbf{x})}, b_i^{(\mathbf{x})} \in \mathbb{F}_2$ s.t for any quadratic function f:

$$f(\mathbf{x}) = \sum_{1 \leq i < j \leq l} a_{i,j}^{(\mathbf{x})} f(\mathbf{x}_i + \mathbf{x}_j) + \sum_{1 \leq i \leq l} b_i^{(\mathbf{x})} f(\mathbf{x}_i)$$

We have assumed that all the elements of S are quadratically independent. For this to hold, the above equations in the symbolic variables $f(\mathbf{x}_i + \mathbf{x}_j)$ and $f(\mathbf{x}_i)$ must be linearly independent. So the number of equations q must be at most the number of variables, which is $\binom{l}{2} + l$. So, we get that:

$$q = |S| \leq \binom{l}{2} + l$$

∎

Lemma 4.10. *For any leaf $v \in L_T$, $l(v) \geq \phi(q(v))$*

Proof. Let $v_0, ..., v_{k-1}, v$ be the path in T from the root to v. Let $\mathbf{x}_i = \mathbf{x}(v_i)$ for $i = 0..k-1$. Apply lemma 4.9 on $\{\mathbf{x}_0, ..., \mathbf{x}_{k-1}\}$ to get that $q(v) \leq \binom{l(v)}{2} + l(v)$. Reversing this formula, since $\phi(x)$ is monotone, we get that $l(v) \geq \phi(q(v))$. ∎

We can now prove our main technical lemma (Lemma 3.1). We start with some technical lemmas. We define $\psi(x)$ to be $x - \phi(x)$ for $x \geq 1$, and 0 for $x < 1$. Notice that ψ is continuous, and $\psi(x) = x - \phi(x)$ for any non-negative integer x. Hence, using Lemma 4.10 we get that:

Lemma 4.11. *For any vertex v in a tree T, $q(v) - l(v) \leq \psi(q(v))$.*

Lemma 4.12. *ψ is increasing and convex.*

Proof. Since ψ is continuous and constant for $x \leq 1$, it's enough to prove the claim for $x > 1$ (for increasing it's clear, and once we've proved ψ is increasing, it shows it's enough to prove convexity for $x > 1$). We first show ψ is increasing.

For $x > 1$, define $y = \phi(x)$, so $x = y^2/2 + y/2$ and $\psi(y) = y^2/2 - y/2$.

$$\frac{d\psi}{dx} = \frac{d\psi}{dy}\frac{dy}{dx} = \frac{\frac{d\psi}{dy}}{\frac{dx}{dy}} = \frac{y - 1/2}{y + 1/2}$$

If $x > 1$ then $y = \phi(x) > 1$, hence $\frac{d\psi}{dx} > 0$ for $x > 1$, and so ψ is increasing.
To show that ψ in convex,

$$\frac{d^2\psi}{dx^2} = \frac{d\left(\frac{y-1/2}{y+1/2}\right)}{dy}\frac{dy}{dx} = \frac{1}{(y+1/2)^3} > 0$$

■

We are now finally ready to prove Lemma 3.1.

Proof. (of Lemma 3.1) We need to prove that any test $\mathbb{T}$ with completeness c and average query complexity $q \geq 1$ accepts a random quadratic function with probability at least $c - 1 + 2^{-\psi(q)}$. Let us mark the probability the test accepts a random quadratic function by p. Let p_T mark the probability that a tree T accepts a random quadratic function. p_T is at least the probability that a random quadratic function reaches a leaf in L_T which is labeled *accept*. So:

$$p_T \geq \sum_{v \in L_T} c(v)2^{-q(v)}$$

We now follow to analyze $p = \mathbb{E}_T[p_T]$.

$$p \geq \mathbb{E}_T[\sum_{v \in L_T} c(v)2^{-q(v)}] = \mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}c(v)2^{-q(v)+l(v)}]$$

.

We divide the sum in the right side into two parts, $p_0 - p_1$, with $p_0, p_1 \geq 0$, where:

$$p_0 = \mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}2^{-q(v)+l(v)}]$$

. and

$$p_1 = \mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}(1 - c(v))2^{-q(v)+l(v)}]$$

.

We start by analyzing p_1. Since for any v always $q(v) \geq l(v)$ we have:

$$p_1 \leq \mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}(1 - c(v))]$$

Recall that by Lemma 4.6 for any tree T we have

$$\sum_{v \in L_T} 2^{-l(v)} = 1$$

and by Lemma 4.3 we have

$$\mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}c(v)] \geq c$$

so we conclude that:

$$p_1 \leq 1 - c$$

We move to analyze p_0. Since $\mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}] = 1$ and since the function $X \to 2^X$ is concave, we have by Jensen's inequality that:

$$p_0 \geq 2^{\mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}(-q(v)+l(v))]}$$

Now, we have that $q(v) - l(v) \leq \psi(q(v))$ by Lemma 4.12, and also by the same lemma, since $q(v) \leq d(v)$, we get $\psi(q(v)) \leq \psi(d(v))$. So we get:

$$\mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}(q(v) - l(v))] \leq \mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}\psi(d(v))]$$

.

Since by Lemma 4.12 ψ is convex, we get that again by Jensen's inequality we get that this is at most $\psi(\mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}d(v)])$. By Lemma 4.7

$$\mathbb{E}_T[\sum_{v \in L_T} 2^{-l(v)}d(v)] \leq q$$

where q is the average query complexity of $\mathbb{T}$. So, we conclude that $p_0 \geq 2^{-\psi(q)}$, and in total

$$p \geq p_0 - p_1 \geq 2^{-\psi(q)} + c - 1$$

■

5. Random quadratic function is far from linear

In this section we prove Lemma 3.2, i.e. that a random quadratic function is far from linear. We will use commonly known facts about quadratic functions.

Any quadratic function can be written as:

$$f(\mathbf{x}) = \mathbf{x}^t A \mathbf{x} + < \mathbf{x}, b >$$

The correlation of f with some linear function g is the g-th Fourier coefficient of f. The Fourier coefficients of quadratic functions are well studied. In particular, it is known that all the Fourier coefficients of f have the same absolute value, and that the number of non-zero Fourier coefficients is $2^{rank(A+A^t)}$. So, in order to show that f has no large correlation with some linear function, it's enough to show that $B = A + A^t$ has high rank. In particular, in order to show that f is $2^{-\Omega(n)}$-far from linear functions, we need to show that B has rank $\Omega(n)$. We will show that the probability that a random quadratic function has rank less than $n/4$ is $2^{-\Omega(n)}$. We will use the following lemma:

Lemma 5.1. *The number of matrices of rank at most k is at most $n^k 2^{nk}$.*

Using Lemma 5.1, it's easy to prove Lemma 3.2. The number of matrices of rank at most $n/4$ is at most $2^{n^2/4(1+o(1))}$. For a random quadratic function, B is a random symmetric matrix with zero diagonal, and so the probability that B has rank less than $n/4$ is $2^{-n^2/4(1+o(1))} = 2^{-\Omega(n)}$.

Now we finish by proving Lemma 5.1.

Proof. Let B be a matrix of rank at most k. There are $\binom{n}{k}$ options to choose k rows which span the row span of the matrix, each other row have at most 2^k options since it must be in the row span of k specific rows. So, the number of possibilities for rank k matrices is at most:

$$\binom{n}{k}(2^k)^{n-k} \leq n^k 2^{nk}$$

■

Acknowledgement

I thank my supervisor, Omer Reingold, for useful comments and for his constant support and interest in the work. I thank Alex Samorodnitsky and Prahladh Harsha for helpful discussions.

References

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Journal of the ACM*, 45(3):501555, 1998. Pre- liminary version in Proc. of FOCS '92.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70122, 1998. Preliminary version in Proc. of FOCS '92.

[BHR05] E. Ben-Sasson, P. Harsha and S. Raskhodnikov. Some 3CNF properties are hard to test. *SIAM Journal on Computing*, 35(1):1–21, 2005. Preliminary Version in Proc. of STOC 2003.

[BCH+96] M. Bellare, D. Coppersmith, J. Hastad, M. Kiwi, and M. Sudan. Linearity testing over characteristic two. *IEEE Transactions on Information Theory*, 42(6):17811795, 1996.

[BLR93] M.Blum, M. Luby and R. Rubinfeld. Self testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549-595, 1993. Preliminary version in Proc. of STOC '90.

[GT05] B. Green and T. Tao. An inverse theorem for the Gowers U3 norm, in *Proceedings of the Edinburgh Mathematical Society*, to appear.

[Sam07] A. Samorodnitsky, Low-degree tests at large distances, In *Proceedings of the 39th Annual ACM Symposium on Theory of computing (STOC 2007)*, pages 506–515, 2007.

[ST00] A. Samorodnitsky and L. Trevisan. A PCP characterization of NP with optimal amortized query complexity. In *Proceedings of the 32nd ACM symposium on Theory of Computation*, pages 191-199, 2000.

[ST06] A. Samorodnitsky and L. Trevisan. Gowers uniformity, influence of variables, and PCPs. In *Proceedings of the 38nd ACM symposium on Theory of Computation*, pages 11-20, 2006.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 527-538
www.stacs-conf.org

AN IMPROVED RANDOMIZED TRUTHFUL MECHANISM FOR SCHEDULING UNRELATED MACHINES

PINYAN LU [1] AND CHANGYUAN YU [1]

[1] Institute for Theoretical Computer Science, Tsinghua University, Beijing, 100084, P.R. China
E-mail address: {lpy,yucy05}@mails.tsinghua.edu.cn
URL: http://www.itcs.tsinghua.edu.cn/{LuPY,YuCY}

Abstract. We study the scheduling problem on unrelated machines in the mechanism design setting. This problem was proposed and studied in the seminal paper of Nisan and Ronen [NR99], where they gave a 1.75-approximation randomized truthful mechanism for the case of two machines. We improve this result by a 1.6737-approximation randomized truthful mechanism. We also generalize our result to a $0.8368m$-approximation mechanism for task scheduling with m machines, which improve the previous best upper bound of $0.875m$[MS07].

1. Introduction

Mechanism design has become an active area of research both in Computer Science and Game Theory. In the mechanism design setting, players are selfish and wish to maximize their own utilities. To deal with the selfishness of the players, a mechanism should both satisfy some game-theoretical requirements such as *truthfulness* and some computational properties such as *good approximation ratio*. The study of their algorithmic aspect was initiated by Nisan and Ronen in their seminal paper "Algorithmic Mechanism Design" [NR99]. The focus of that paper was on the scheduling problem on unrelated machines, for which the standard mechanism design tools (VCG mechanisms [Clarke71, Groves1973, Vickrey61])do not suffice. They proved that no deterministic mechanism can have an approximation ratio better than 2 for this problem. This bound is tight for the case of two machines. However if we allow randomized mechanisms, this bound can be beaten. In particular they gave a 1.75-approximation randomized truthful mechanism for the case of two machines. Since then, many researchers have studied the scheduling problem on unrelated machines in mechanism design setting [JP99, Sourd01, SS02, SX02, GMW07, CKV07, CKK07, MS07]. However their mechanism remains the best to the best of our knowledge. In a recent paper [MS07], Mu'alem and Schapira proved a lower bound of 1.5 for this setting. So to explore the exact bound between 1.5 and 1.75 is an interesting open problem in this area. In this paper,

1998 ACM Subject Classification: algorithmic mechanism design.
Key words and phrases: truthful mechanism, scheduling.
Supported by the National Natural Science Foundation of China Grant 60553001 and the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

we improve the upper bound from 1.75 to 1.6737. Formally we give a 1.6737-approximation randomized truthful mechanism for task scheduling with two machines. Using similar techniques of [MS07], we also generalize our result to a $0.8368m$-approximation mechanism for task scheduling with m machines.

Let us describe the problem more carefully. There are m machines and n tasks, and each machine is controlled by an agent. We use t_j^i to denote the running time of task j on machine i, which is also called the type value of the agent(machine) i on task j. The objective is to minimize the completion time of the last assignment (the *makespan*). Unlike in the classical optimization problem, the scheduling designer does not know t_j^i. Each selfish agent i holds his/her own type values (the t_j^is). In order to motivate the agents to report their true value t_j^is, the mechanism needs to pay the agents. So a mechanism consists of an allocation algorithm and a payment algorithm. A mechanism is called *truthful* when telling one's true value is among the optimal strategies for each agent, no matter how other agents behave. Here the utility of each agent is the payment he/she gets minus the load of tasks allocated to his/her machine. When randomness is involved, there are two versions of truthfulness: in the stronger version, i.e. *universally truthfulness*, the mechanism remains truthful even if the agents know the random bits; in the weaker version, i.e. *truthfulness in expectation*, an agent maximizes his/her expected utility by telling the true type value. Our mechanisms proposed in this paper are universally truthful.

Now we can talk about the high level idea of the technical part. Here we only talk about the allocation algorithms, and the corresponding payment algorithms, which make the mechanism truthful, will be given later. First we describe Nisan and Ronen's mechanism [NR99]. In their mechanism, each task is allocated independently. For a particular task j, if the two values t_j^1 and t_j^2 are relatively close to each other, say $t_j^1/t_j^2 \in [3/4, 4/3]$, then they allocate task j randomly to machine 1 or 2 with equal probability; if one is much higher then the other, say $t_j^1/t_j^2 > 4/3$ or $t_j^2/t_j^1 > 4/3$, the task j is allocated to the more efficient machine. The main idea of our mechanism is to partition the tasks into three categories rather than two. So we need two threshold values, say α, β, where $1 < \beta < \alpha$ and a biased probability r , where $1/2 < r < 1$. If the two values are relatively close to each other, say $t_j^1/t_j^2 \in [1/\beta, \beta]$, or one is much higher then the other, say $t_j^1/t_j^2 > \alpha$ or $t_j^2/t_j^1 > \alpha$, we do the same things as Nisan and Ronen's mechanism. In the remaining case, one is significantly larger than the other, but however still does not dominate, say $t_j^1/t_j^2 \in [\beta, \alpha]$ or $t_j^2/t_j^1 \in [\beta, \alpha]$. In this case, we allocate the task j to the more efficient one with a higher probability r ($r > 1/2$) and the less efficient one with a lower probability $1 - r$. The mechanism is quite simple, so it is very computationally efficient. Intuitively our mechanism will give better approximation ratios by choosing suitable parameters α, β and r. This is indeed true. We can prove an improved approximation ratio of 1.6737 by choosing $\alpha = 1.4844, \beta = 1.1854, r = 0.7932$. However, the proof is quite involved. One reason is that the situation for the new case (middle case) is more complicated than the original two. The main reason is that their approach becomes infeasible in the analysis of our mechanism. The proof in Nisan and Ronen's paper is basically case by case, but unfortunately the number of subcases increases double exponentially with the number of task types. So we introduce some substantial new proof techniques to overcome this. We also think this techniques may further improve the upper bound.

1.1. Related Work

Scheduling on unrelated machines is one of the most fundamental scheduling Problems. For this NP-hard optimization problem, there is a polynomial time algorithm with approximation ratio of 2[LST87]. Especially if the number of machines is bounded by some constant, Angel, Bampis and Kononov gave an FPTAS[ABK01]. However there is no corresponding payment strategy to make either of the above allocation algorithms truthful.

The study of this problem in the mechanism design setting is initiated by Nisan and Ronen. In their paper [NR99], they gave a 1.75-approximation randomized truthful mechanism for two machines. This result was generalized by Mu'alem and Schapira to a $0.875m$-approximation randomized mechanism for m machines[MS07]. We improved the two upper bounds to 1.6737 and $0.8368m$ respectively.

For the lower bound side, Nisan and Ronen gave a lower bound of 2 for deterministic version. This bound was improved by Christodoulou, Koutsoupias and Vidali to $1+\sqrt{2}$ for 3 or more machines [CKV07]. For the randomized version, Mu'alem and Schapira gave a lower bound of $2-1/m$[MS07]. This also holds for the weaker notion of truthfulness, i.e., truthfulness in expectation.

Lavi and Swarmy considered a restricted variant, where each task j only has two values of running time , and gave a 3-approximation randomized truthful mechanism [LS07]. They first use the cycle monotonicity in designing mechanisms.

In [CKK07], Christodoulou, Koutsoupias and Kovács considered the fractional version of this problem, in which each task can be split among the machines. For this version, they gave a lower bound of $2-1/m$ and an upper bound of $(m+1)/2$. We remark that these two bounds are closed for the case of two machines as in the integral deterministic version. So to explore the exact bound for the randomized version seems very interesting and desirable. We believe that our work in this paper is an important step toward this objective.

2. Problem and Definitions

In this section we review some definitions and results on mechanism design and scheduling problem. More details can be found in[NR99].

In a mechanism design problem, there are usually some resources to distribute among n agents. Every agent i has a type value t^i, which denotes his/her preference on the resources. Let $t=(t^i)_{i\in[m]}$ denote the vector of all agents' type values and t^{-i} denote the vector of all agents' type vectors except agent i's. Receiving all the type values t from agents, the mechanism will produce an output $o(t)=(x(t),p(t))$. Here $x(t)$ specifies the allocation of the resources and is produced by an allocation algorithm. $p(t)$ specifies the payment to agents and is produced by an payment algorithm. Every agent i has a valuation $v^i(x,t^i)$, which describe his/her preference on the output allocation. The agent i's objective is maximizing his/her utility function u^i, where $u^i=v^i+p^i$, and p^i is the payment obtained from the mechanism. The mechanism's objective is to maximize an objective function $g(o,t)$. Formally, we have the following definitions.

Definition 2.1. A mechanism is a pair of Algorithms $M=(X,P)$.

- *Allocation Algorithm X:* Its input is m agents' type vectors, $t^1,t^2,\cdots,t^m$, which are reported by the agents. and its output is $x=(x^1,\cdots,x^m)$, where $x^i=(x^i_j)_{j\in[n]}\in\{0,1\}^n$ are allocation vector of agent i.

- *Payment Algorithm P:* It outputs a payment vector $p = (p_1, \cdots, p_m)$, which depends both on agents' type vectors and allocation vectors produced by allocation algorithm.

A mechanism is *deterministic* if both the allocation algorithm and payment algorithm are deterministic. When at least one of them uses random bits, it is called a *randomized mechanism*.

In order to increase utility, an agent may lie when reporting his/her type values. But for some mechanisms, no agent can increase his/her utility by lying. This nice property of a mechanism is called *truthfulness*. We give the formal definitions of truthfulness.

Definition 2.2. A deterministic mechanism is **truthful** iff for every agent, reporting his/her true type values is among the best strategies to maximize his/her utility, no matter how other agents acts. A randomized mechanism is **truthful in expectation** iff no agent can increase his/her expected utility by lying. A randomized mechanism is **universally truthful** iff it remains truthful even if the agents know the random bits.

From now on, we will only focus on truthful mechanisms. The most important positive result in mechanism design is generalized Vickrey-Clarke-Groves(VCG) mechanism [Vickrey61, Groves1973, Clarke71]. Many known truthful mechanisms are all in VCG family. The mechanisms of VCG family usually apply to mechanism design problem in which the objective function is the (weighted) sum of all agents' valuations. To be formal, we have

Definition 2.3. [NR99] A mechanism $M = (X, P)$ belongs to weighted VCG family if there are real numbers(weights) $\beta^1, \cdots, \beta^n > 0$, such that:

(1) the problem's objective function satisfies $g(o,t) = \sum_i \beta^i v^i(t^i, o)$.
(2) $o(t) \in argmax_o(g(o,t))$.
(3) $p^i(t) = \frac{1}{\beta^i} \sum_{i' \neq i} \beta^{i'} v^{i'}(t^{i'}, o(t)) + h^i(t^{-i})$, where $h^i()$ is an arbitrary function of t^{-i}.

Theorem 2.4. *(*[Roberts79]*) A weighted VCG mechanism is truthful.*

Now we specify these mechanism notions in the problem of scheduling unrelated machines. Assume there are n tasks to be allocated to m machines, each of which is controlled by an agent. Each agent i's type value is $t^i = (t^i_j)_{j \in [n]}$, where t^i_j denotes the time to perform task j on machine i.

We use a binary array $x^i = (x^i_j)_{j \in [n]}$ to specify the allocation of tasks to machine i. x^i_j is 1 if task j is allocated to machine i and otherwise 0. Let $x = (x^i)_{i \in [m]}$ denote the allocation of all the tasks. For an allocation x, agent i's valuation is $v^i = -x^i \cdot t^i$, where $x^i \cdot t^i = \sum_{j=1}^n x^i_j t^i_j$.

Definition 2.5. Given any allocation x of the tasks, the longest running time of the machines is called the makespan of the allocation. Formally, $\text{makespan}(x) = max_{i \in [m]} x^i \cdot t^i$.

The objective of the mechanism is to minimize the (expected) makespan of the allocation. This is not the (weighted) sum of all agents' valuations. So we can not apply VCG mechanism here. However, we remark that if there is only one task, the makespan can be viewed as the sum of all agents' valuations. We will use this observation in our analysis.

From [NR99] and [MS07], we know that there is no optimal truthful mechanism for this problem, even if we allow super-polynomial running time and randomness. So we will try to find a truthful mechanism with good approximation ratio.

Definition 2.6. Let $t_M(t)$ be the (expected) makespan of the mechanism M on instances t and $t_{opt}(t)$ be the optimal makespan of instance t. We say mechanism M has approximation ratio c iff for any instance t, $t_M(t)/t_{opt}(t) \leq c$.

3. Our Mechanism and the Analysis

In this section, we give a truthful scheduling mechanism for 2 machines case, and show that its approximation ratio is 1.6737. Then we generalize our result to the m machines case as in [MS07] and obtain a $0.8368m$-approximation randomized truthful mechanism.

3.1. Generalized Randomly Biased Mechanism

Parameters: Real numbers $\alpha > \beta \geq 1 > r \geq \frac{1}{2}$.
(Here we choose $\alpha = 1.4844, \beta = 1.1854, r = 0.7932$.)
Input: The reported type vectors $t = (t^1, t^2)$.
Output: A randomized allocation $x = (x^1, x^2)$,
and a payment $p = (p^1, p^2)$.
Allocation and Payment algorithm:
$x_j^1 \leftarrow 0, x_j^2 \leftarrow 0, j = 1, 2 \cdots, n; p^1 \leftarrow 0; p^2 \leftarrow 0$.
For each task $j = 1, 2 \cdots, n$ do

$$s_j \leftarrow \begin{cases} \alpha, & \text{with probability } 1 - r, \\ \beta, & \text{with probability } r - 1/2, \\ 1/\beta, & \text{with probability } r - 1/2, \\ 1/\alpha, & \text{with probability } 1 - r. \end{cases}$$

if $t_j^1 < s_j t_j^2$,
$x_j^1 = 1, p^1 \leftarrow p^1 + s_j t_j^2$;
else
$x_j^2 = 1, p^2 \leftarrow p^2 + s_j^{-1} t_j^1$.

Theorem 3.1. *The Generalized Randomly Biased Mechanism (GBM for short) is universally truthful and can achieve a* 1.6737*-approximation solution for task scheduling with two machines.*

We will prove this theorem in the following two subsections. In 3.2, we will prove that our mechanism is universally truthful. Then we analyze its approximation ratio in 3.3

3.2. Truthfulness

Lemma 3.2. *The Generalized Randomly Biased Mechanism is universally truthful.*

Proof. To prove that the GBM is universally truthful, we only need to prove that it is truthful when the random sequence s_j is fixed. Since the utility of an agent equals the sum of the utilities obtained from each task and our mechanism is task-independent, we only need consider the case of one task. In this case, say s_j is fixed and there is only one task j, the mechanism is exactly the VCG mechanism with weight $(1, s_j)$. Since a weighted VCG is truthful, the GBM mechanism is universally truthful.

3.3. Estimation of the Approximation Ratio

If this subsection, we will estimate the approximation ratio of our GBM mechanism. Since we already proved that GBM is universally truthful in 3.2, we only need to focus on the allocation algorithms of GBM. So we can restate the allocation algorithms for GBM in an equivalent but more understandable way. Intuitively we should assign one task with larger probability to the machine which has smaller type value(running time) on it. The idea of our mechanism is to partition all the tasks into several types according to the ratio of two agents' type values. For different types of tasks, we use different biased probabilities to allocate them. To be formal, we have the following definition.

Definition 3.3. For a task j, we call it an h-task iff $\frac{t_j^i}{t_j^{3-i}} > \alpha$ for some $i \in \{1,2\}$; we called it an m-task iff $\beta < \frac{t_j^i}{t_j^{3-i}} \leq \alpha$ for some $i \in \{1,2\}$; we call it an l-task if $\frac{t_j^i}{t_j^{3-i}} \leq \beta$ for any $i \in \{1,2\}$.

Then, we have the following claim.

Claim 3.4. *The GBM mechanism allocates the tasks in the same way as the following allocating algorithm does.*

- *For h-task, we allocate it to the machine with lower type value.*
- *For m-task, we allocate it to the more efficient machine with probability r and to the less efficient machine with probability $1-r$.*
- *For l-task, we allocate it to two machines with equal probabilities.*

Proof. For each task j, we consider the probability that it is allocated to machine 1 in GBM. According to the ratio of $\frac{t_j^1}{t_j^2}$, we have the following 5 cases:

- *Case 1:* $t_j^1 \geq \alpha t_j^2$, then $Pr(x_j^1 = 1) = 0$
- *Case 2:* $\beta \leq t_j^1 < \alpha t_j^2$, then $Pr(x_j^1 = 1) = 1 - r$
- *Case 3:* $\beta^{-1} \leq t_j^1 < \beta t_j^2$, then $Pr(x_j^1 = 1) = (1-r) + (r - \frac{1}{2}) = \frac{1}{2}$
- *Case 4:* $\alpha^{-1} \leq t_j^1 < \beta^{-1} t_j^2$, then $Pr(x_j^1 = 1) = (1-r) + (r - \frac{1}{2}) + (r - \frac{1}{2}) = r$
- *Case 5:* $t_j^2 < \alpha^{-1} t_j^2$, then $Pr(x_j^1 = 1) = (1-r) + (r - \frac{1}{2}) + (r - \frac{1}{2}) + (1-r) = 1$

The probabilities that task j is assigned to machine 1 by two algorithms are always the same, so the lemma is true. ∎

Remark 3.5. This claim only says that the (distribution of) allocation produced by the two methods are the same. However if we use this allocation algorithm stated in the claim, we can only make the mechanism truthful in expectation.

As in [NR99], we obtain the following crucial claim, which can help us cut the number of tasks. The proof of this claim is similar , and we put it in the Appendix.

Claim 3.6. *To analyze the performance of the generalized randomized biased mechanism, we only need consider the following cases:*

(1) *For each h-task j, the ratio of the two machines' type value is arbitrarily close to α. So we can assume it equals α.*
(2) *If OPT allocates an l-task j to machine i, then $t_j^{3-i}/t_j^i = \beta$.*
(3) *If OPT allocates an m-task j to machine i which has smaller type value, then $t_j^{3-i}/t_j^i = \alpha$.*

(4) *If OPT allocates an m-task j to machine i which has bigger type value, then* $t_j^{3-i}/t_j^i = \beta^{-1}$.
(5) *One of the machines is more efficient than the other on all h-tasks. We assume it's machine* 1.
(6) *There are at most* 8 *tasks* A, B, C, D, E, F, G, H. *In OPT, tasks* A, C, E, G *are allocated to machine* 1, *and the others to machine* 2. *Tasks* A, B *are h-tasks. Tasks* C, D, E, F *are m-tasks and tasks* G, H *are l-tasks.*

From the above analysis, we know that we only need to consider the reduced case as described in Figure 1.

type	task	t_j^1	t_j^2	opt-alloc	gbm-alloc(probability)
h_1	A	a	αa	1	$1:0$
h_2	B	b	αb	2	$1:0$
m_1^1	C	c	αc	1	$r:(1-r)$
m_2^1	D	d	βd	2	$r:(1-r)$
m_1^2	E	βe	e	1	$(1-r):r$
m_2^2	F	αf	f	2	$(1-r):r$
l_1	G	g	βg	1	$\frac{1}{2}:\frac{1}{2}$
l_2	H	βh	h	2	$\frac{1}{2}:\frac{1}{2}$

Figure 1: The Reduced Case.

Now we can estimate the approximation ratio based on this reduced case.

Lemma 3.7. *The allocation produced by* ***GBM*** *is a* 1.6737*-approximation solution for the task scheduling problem with two machines.*

Proof. Let t_{opt} be the make-span of an optimal solution and let t_{gbm} be the expected makespan of allocations produced by GBM. We want to show that $t_{gbm} \leq 1.6737 t_{opt}$.

From the allocation of the optimal solution, we have that

$$t_{opt} = max\{a + c + \beta e + g, \alpha b + \beta d + f + h\}.$$

Now we will estimate the expected makespan of our mechanism t_{gbm}. First we introduce some notation which will be used in the following analysis. We will treat the same name X $(X = A, B, \cdots, H)$ as a random variable, which denotes the assignment of the task X. For example, $C = 2$ means that our mechanism assigns the task C to the second machine. Then the last column in Figure 1 can also be viewed as the distribution of the random variable X $(X = A, B, \cdots, H)$. For example $Pr(C = 1) = r$ and $Pr(C = 2) = 1 - r$. Since our mechanism assigns each task independently, the random variables are also independent of each other. More precisely, for any $X, Y \in \{A, B, \cdots, H\}$, $i, j \in \{1, 2\}$ and $X \neq Y$, we have

$$Pr(X = i, Y = j) = Pr(X = i)Pr(Y = j).$$

We use a random variable M to denote the machine finishing last. More precisely, $M = 1$ means the completion time of the first machine is not earlier than the second machine, otherwise we have $M = 2$.

Now we compute the contribution of each task to t_{gbm}. Let the j-th task be X. Then its contribution to t_{gbm} contains two parts. First part is from t_j^1. t_j^1 contributes to t_{gbm}

iff our mechanism assigns task X to 1 (e.t. $X = 1$) and the machine 1 finishes later (e.t. $M = 1$). The situation for t_j^2 is similar. To sum up, the contribution of the j-th task X to t_{gbm} is

$$Pr(M = 1, X = 1)t_j^1 + Pr(M = 2, X = 2)t_j^2.$$

For example, the contribution of task C to t_{gbm} is

$$Pr(M = 1, C = 1)c + \alpha Pr(M = 2, C = 2)c = (Pr(M = 1, C = 1) + \alpha Pr(M = 2, C = 2))c.$$

Similarly, we can compute the contribution of each task to t_{gbm} easily. To simplify the notation, we use C_x $(x = a, b, \cdots, h)$ to denote the coefficient of x in t_{gbm}. So we have

$$t_{gbm} = C_a a + C_b b + C_c c + C_d d + C_e e + C_f f + C_g g + C_h h.$$

where

$$\begin{aligned}
C_a &= Pr(M = 1), \\
C_b &= Pr(M = 1), \\
C_c &= Pr(M = 1, C = 1) + \alpha Pr(M = 2, C = 2), \\
C_d &= Pr(M = 1, D = 1) + \beta Pr(M = 2, D = 2), \\
C_e &= \beta Pr(M = 1, E = 1) + Pr(M = 2, E = 2), \\
C_f &= \alpha Pr(M = 1, F = 1) + Pr(M = 2, F = 2), \\
C_g &= Pr(M = 1, G = 1) + \beta Pr(M = 2, G = 2), \\
C_h &= \beta Pr(M = 1, H = 1) + Pr(M = 2, H = 2).
\end{aligned}$$

Since

$$\begin{aligned}
t_{gbm} &= C_a a + C_b b + C_c c + C_d d + C_e e + C_f f + C_g g + C_h h \\
&= (C_a a + C_c c + \frac{C_e}{\beta}\beta e + C_g g) + (\frac{C_b}{\alpha}\alpha b + \frac{C_d}{\beta}\beta d + C_f f + C_h h) \\
&\leq max(C_a, C_c, \frac{C_e}{\beta}, C_g)(a + c + \beta e + g) + max(\frac{C_b}{\alpha}, \frac{C_d}{\beta}, C_f, C_h)(\alpha b + \beta d + f + h) \\
&\leq max(C_a, C_c, \frac{C_e}{\beta}, C_g)t_{opt} + max(\frac{C_b}{\alpha}, \frac{C_d}{\beta}, C_f, C_h)t_{opt}
\end{aligned}$$

So the performance of our mechanism is bounded by

$$max(C_a, C_c, \frac{C_e}{\beta}, C_g) + max(\frac{C_b}{\alpha}, \frac{C_d}{\beta}, C_f, C_h).$$

We will give bound for every possible sum between $\{C_a, C_c, \frac{C_e}{\beta}, C_g\}$ and $\{\frac{C_b}{\alpha}, \frac{C_d}{\beta}, C_f, C_h\}$. First

$$C_a + \frac{C_b}{\alpha} = Pr(M = 1) + \frac{Pr(M = 1)}{\alpha} \leq 1 + \frac{1}{\alpha}.$$

So $C_a + \frac{C_b}{\alpha}$ is bounded by $1 + \frac{1}{\alpha}$. Later we will choose suitable parameter α so that this value is not too big.

Now we analyze a more complicated case, say $C_c + C_f$. Substituting C_c and C_f, we have

$$C_c + C_f = Pr(M = 1, C = 1) + \alpha Pr(M = 2, C = 2) + \alpha Pr(M = 1, F = 1) + Pr(M = 2, F = 2).$$

Here we use P_{ijk} to denote the joint distribution of three random variables M, C, F (e.t. $P_{ijk} = Pr(M = i, C = j, F = k)$). Then we can rewrite the formula as following

$$P_{111} + P_{112} + \alpha(P_{221} + P_{222}) + \alpha(P_{111} + P_{121}) + (P_{212} + P_{222}).$$

Then we recombine the terms as following

$$(P_{111} + P_{112} + P_{212}) + \alpha(P_{111} + P_{121} + P_{221}) + (1 + \alpha)P_{222}.$$

The first term is bounded by $Pr(C = 1)$ (since $Pr(C = 1) = P_{111} + P_{112} + P_{212} + P_{211}$); similarly the second term is bounded by $\alpha Pr(F = 1)$; the third term is bounded by $(1 + \alpha)Pr(C = 2, F = 2)$. So we can bound $C_c + C_f$ by

$$\begin{aligned} &Pr(C = 1) + \alpha Pr(F = 1) + (1 + \alpha)Pr(C = 2, F = 2) \\ &= r + \alpha(1 - r) + (1 + \alpha)r(1 - r) = 2r + \alpha - r^2 - \alpha r^2. \end{aligned}$$

Similarly we can bound the remaining 14 sums as follows. Some of proofs are slightly more complicated but all of them are along similar lines. We only list the bounds here, and the details are omitted here due to the space limitation.

(1) $C_a + \frac{C_d}{\beta} \le 1 + \frac{r}{\beta}$.
(2) $C_a + C_f \le 1 + (1 - r)\alpha$.
(3) $C_a + C_h \le 1 + \frac{\beta}{2}$.
(4) $C_c + \frac{C_b}{\alpha} \le 1 + \frac{1}{\alpha}$. (Here we use the assumption that $\alpha \le 1 + \frac{1}{\alpha}$.)
(5) $C_c + \frac{C_d}{\beta} \le \frac{r^2}{\beta} + 1 + r^2 + \alpha - r + \alpha r$.
(6) $C_c + C_h \le \frac{1}{2} + \frac{1}{2}r + \alpha - \alpha r + \frac{1}{2}\beta r$.
(7) $\frac{C_e}{\beta} + \frac{C_b}{\alpha} \le 1 + \frac{1}{\alpha}$.
(8) $\frac{C_e}{\beta} + \frac{C_d}{\beta} \le (1 - r) + \frac{1}{\beta}r + (1 + \frac{1}{\beta})r(1 - r) \le C_c + C_f$.
(9) $\frac{C_e}{\beta} + C_f \le \frac{r^2}{\beta} + 1 + r^2 + \alpha - r + \alpha r$.
(10) $\frac{C_e}{\beta} + C_h \le 1 + \frac{r}{2\beta} + \frac{1}{2}\beta - \frac{1}{2}r$.
(11) $C_g + \frac{C_b}{\alpha} \le 1 + \frac{1}{\alpha}$. (Here we use the assumption that $\alpha \le 1 + \frac{1}{\alpha}$.)
(12) $C_g + \frac{C_d}{\beta} \le 1 + \frac{r}{2\beta} + \frac{1}{2}\beta - \frac{1}{2}r$.
(13) $C_g + C_f \le \frac{1}{2} + \frac{1}{2}r + \alpha - \alpha r + \frac{1}{2}\beta r$.
(14) $C_g + C_h \le \frac{3}{4} + \frac{3}{4}\beta$.

To sum up, we have 9 different bounds: $1+\frac{1}{\alpha}$, $2r+\alpha-r^2-\alpha r^2$, $1+\frac{r}{\beta}$, $1+(1-r)\alpha$, $1+\frac{\beta}{2}$, $\frac{r^2}{\beta}+1+r^2+\alpha-r+\alpha r$, $\frac{1}{2}+\frac{1}{2}r+\alpha-\alpha r+\frac{1}{2}\beta r$, $1+\frac{r}{2\beta}+\frac{1}{2}\beta-\frac{1}{2}r$, $\frac{3}{4}+\frac{3}{4}\beta$, and one assumption that $\alpha \le 1 + \frac{1}{\alpha}$. We want to choose suitable parameter α, β, r such that the assumption is satisfied and the maximal bound is as small as possible. This can be easily done numerically by a mathematical tool such as Matlab. We can choose $\alpha = 1.4844, \beta = 1.1854, r = 0.7932$. Substituting these values, we can verify that all the bounds are less than 1.6737. So we proved that our mechanism has an approximate ratio of 1.6737. ∎

3.4. An Improved Mechanism for m Machines

As an application of our main result, we turn to the case of m machines. In [NR99], Nisan and Ronen gave a truthful deterministic mechanism that achieves an m-approximation. Recently, Mu'alem and Schapira [MS07] generalized Nisan and Ronen's truthful randomized mechanism for 2 machines to the case of m machines. They partitioned the m machines into

two sets of machines with equal size, S_1 and S_2. Then they construct a new instance with only two machines, with type values $t^i_j = min_{a\in S_i} t^a_j, i = 1, 2$. Applying the mechanism for 2 machines case, They showed a universally truthful randomized mechanism that obtains an approximation of $0.875m$. Using this idea and our improved result for two machines case, we can improve the ratio from $0.875m$ to $0.8368m$. To be self contained, we give the formal description of the mechanism here. The proof is similar with [MS07] and omitted here.

Parameters: real numbers $\alpha > \beta \geq 1 > r \geq \frac{1}{2}$.
Input: the reported type value vectors $t = (t^1, t^2, \cdots, t^m)$.
Output: an randomized allocation $x = (x^1, x^2, \cdots, x^m)$ and a payment $p = (p^1, p^2, \cdots, p^m)$.
Mechanism:

(1) For each machine i, let $x^i \leftarrow \emptyset; p^i \leftarrow 0$.
(2) Partition the set of machines into two sets S_1, S_2 with equal size. If m is not even, we can add an extra machine with infinite type values on every task.
(3) For each task j, Let $t^a = min_{i\in S_1} t^i_j$, $a = argmin_{i\in S_1} t^i_j$, $t^{a'} = min_{i\in S_1-\{a\}} t^i_j$. Let $t^b = min_{i\in S_2} t^i_j$, $b = argmin_{i\in S_1} t^i_j$, $t^{b'} = min_{i\in S_1-\{a\}} t^i_j$.
(4) Apply our mechanism GBM for two machines case to machine a and b on task j. Also the payment strategy need a little change. If a gets the task, and it will gain a payment p^a_j in GBM, then we pay it $min\{p^a_j, t^{a'}_j\}$. If b gets the task, and it will gain a payment p^b_j in GBM, then we pay it $min\{p^b_j, t^{b'}_j\}$. This change is in order to keep the mechanism truthful.

Theorem 3.8. *m-GBM is an universally truthful randomized mechanism for the scheduling problem that obtains an approximation ratio of* $0.8369m$ *when choosing* $\alpha = 1.4844, \beta = 1.1854, r = 0.7932$.

4. Conclusions and Open Problems

This is the first improvement since Nisan and Ronen proposed the problem and the 1.75-mechanism. We believe it is possible to further improve the upper bound using our technics. A direct open problem is to close the gap between the lower bound of 1.5 and our new upper bound of 1.6737.

Another more important direction is to generalize the mechanisms for 2 machine to mechanisms for m machines in a more clever way. In the general case, the gap between the best lower bounds (constants) and the best upper bounds ($\Theta(m)$) is huge both in deterministic and randomized versions. Any improvement in either direction is highly desirable.

References

[ABK01] Angel, E., Bampis, E. and Kononov, A. A FPTAS for Approximating the Unrelated Parallel Machines Scheduling Problem with Costs. *In Proceedings of the 9th Annual European Symposium on Algorithms* (August 28 – 31, 2001). F. M. Heide, Ed. *Lecture Notes In Computer Science*, vol. 2161. Springer-Verlag, London, 194 – 205.

[CKK07] Christodoulou, G., Koutsoupias, E. and Kovács, A. Mechanism Design for Fractional Scheduling on Unrelated Machines *In Proceedings of ICALP* 2007, 40 – 52.

[CKV07] Christodoulou, G., Koutsoupias, E. and Vidali, A. A lower bound for scheduling mechanisms. *In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (SODA '07), 2007, 1163 − 1170.

[Clarke71] Clarke, E. H. Multipart Pricing of Public Goods. *Public Choice*, 11 : 17 − 33, 1971.

[GMW07] Gairing, M., Monien, B., and Woclaw, A. A faster combinatorial approximation algorithm for scheduling unrelated parallel machines.*Theor. Comput. Sci.* 380, 1 − 2 (Jun. 2007), 87 − 99.

[Groves1973] Groves, T. Incentives in Teams. *Econometrica*, 41 : 617 − 631, 1973.

[JP99] Jansen, K. and Porkolab, L. Improved approximation schemes for scheduling unrelated parallel machines.*In Proceedings of the Thirty-First Annual ACM Symposium on theory of Computing* (Atlanta, Georgia, United States, May 01 − 04, 1999). *STOC '99. ACM Press*, New York, NY, 408 − 417.

[LS07] Lavi, R. and Swamy, C. Truthful mechanism design for multi-dimensional scheduling via cycle-monotonicity. In *Proceedings 8th ACM Conference on Electronic Commerce (EC)*, 2007. 252-261 .

[LST87] Lenstra, J. K., Shmoys, D. B., and Tardos, É. Approximation algorithms for scheduling unrelated parallel machines.*Math. Program.* 46, 3 (Apr. 1990), 259 − 271.

[MS07] Mu'alem, A. and Schapira, M. Setting lower bounds on truthfulness. *In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (New Orleans, Louisiana, January 07 − 09, 2007). *Symposium on Discrete Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1143 − 1152.

[NR99] Nisan, N. and Ronen, A. Algorithmic mechanism design (extended abstract). *In Proceedings of the Thirty-First Annual ACM Symposium on theory of Computing* (Atlanta, Georgia, United States, May 01 − 04, 1999). *STOC '99. ACM Press*, New York, NY, 129 − 140.

[Roberts79] Roberts, K. The characterization of implementable choise rules. *In Jean-Jacques Laffont, editor, Aggregation and Revelation of Preferences*, pages 321C349. North-Holland, 1979. Papers presented at the first European Summer Workshop of the Econometric Society.

[Sourd01] Sourd, F. Scheduling Tasks on Unrelated Machines: Large Neighborhood Improvement Procedures.*Journal of Heuristics* 7, 6 (Nov. 2001), 519 − 531.

[SS02] Schulz, A. S. and Skutella, M. Scheduling Unrelated Machines by Randomized Rounding. *SIAM J. Discret. Math.* 15, 4 (Apr. 2002), 450 − 469.

[ST93] Shmoys, D. B. and Tardos, É. Scheduling unrelated machines with costs. *In Proc. Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (Austin, Texas, United States, 1993). *Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics*, Philadelphia, PA, 448 − 454.

[SX02] Serna, M. and Xhafa, F. Approximating Scheduling Unrelated Parallel Machines in Parallel. *Comput. Optim. Appl.* 21, 3 (Mar. 2002), 325 − 338.

[Vickrey61] Vickrey, W. Counterspeculation, Auctions, and Competitive Sealed Tenders. *Journal of Finance*, 16 : 8 − 37, 1961.

Appendix

Proof of Claim 3.6

(1) For h-task j, assume $t_j^1 < t_j^2$. We can decrease t_j^2 to αt_j^1, then t_{gbm} will not change since GBM always allocates task j to agent 1. But this may help OPT, so the approximation ratio can only be worse.
(2) Increasing t_j^{3-i} to βt_j^i will not affect OPT but will increase t_{gbm}. This is because the probability to allocate j does not change as long as it is still an l-task, and one type value is increased.
(3) It is similar with the above. We can keep increasing t_j^{3-i} while j is still m-task. Here $\beta \le t_j^{3-i}/t_j^i \le \alpha$, so we can make it equal α.
(4) Here $\beta^{-1} \ge t_j^{3-i}/t_j^i \ge \alpha^{-1}$, so we can increase t_j^{3-i} until this ratio equals β^{-1}.
(5) This is the same as in [NR99]. We omit the proof here.
(6) Let $h_a, l_a, a \in \{1,2\}$ denote an h-task or l-task respectively which is allocated to agent a in OPT. Let $m_b^a, a, b \in \{1,2\}$ denote an m-task allocated to agent b in OPT, on which

agent a has smaller type value. So there are 8 types of tasks. We will prove that any two task j_1, j_2 of the same type can be combined into a single task j of the same type. Firstly, notice that j_1, j_2 have the same ratio of the two agents' type values. so task j still has this ratio, hence the same type. Further more, they are all allocated by GBM with the same probability distribution.

In one direction, combining will leave t_{opt} unchanged. Obviously, combining can only increase t_{opt} because any allocation obtained for the new instance can be get for the old one. Also t_{opt} can be achieved for the new instance since two tasks of the same type are allocated to the same agent.

In the other direction, combining can only increase t_{gbm}.
For the h-task case, t_{gbm} is also unchanged because GBM always allocate the h-tasks to the more efficient agent.

For the m-task case, assume j_1, j_2 are both $m_b^a, a, b \in \{1, 2\}$. Let Y denote an allocation of all the tasks except task j_1, j_2. Let t_{Y,j_1,j_2} (resp.$t_{Y,j}$) denote the expected make-span when j_1, j_2 (resp. j) are (is) allocated by GBM and all other tasks are allocated according to Y. We have to show that $t_{Y,j_1,j_2} \leq t_{Y,j}$. Let T^1, T^2 denote finishing time of two agents respectively when allocation is Y.
If agent i finishes last regardless of how j_1, j_2 are allocated, then

$$t_{Y,j_1,j_2} = T^i + r_i(t^i_{j_1} + t^i_{j_2}) = t_{Y,j}$$

Here r_i denotes the probability that j_1, j_2 and j are allocated to agent i. Otherwise, if agent i finishes last iff both j_1, j_2 are allocated to it, then $T^{3-i} \leq T^i + t^i_{j_1} + t^i_{j_2}$

$$\begin{aligned}
&t_{Y,j_1,j_2}\\
&= r_i^2(T^i + t^i_{j_1} + t^i_{j_2}) + r_i(1-r_i)(T^{3-i} + t^{3-i}_{j_1} + T^{3-i} + t^{3-i}_{j_2}) + (1-r_i)^2(T^{3-i} + t^{3-i}_{j_1} + t^{3-i}_{j_2})\\
&\leq (r_i^2 + r_i(1-r_i))(T^i + t^i_{j_1} + t^i_{j_2}) + ((1-r_i)^2 + r_i(1-r_i))(T^{3-i} + t^{3-i}_{j_1} + t^{3-i}_{j_2})\\
&= r_i(T^i + t^i_{j_1} + t^i_{j_2}) + (1-r_i)(T^{3-i} + t^{3-i}_{j_1} + t^{3-i}_{j_2})\\
&= t_{Y,j}
\end{aligned}$$

Finally assume that $t^i_{j_1} \geq t^i_{j_2}, i = 1, 2$ and consider the last case where the agent to which j_1 is allocated finishes last. In this case

$$\begin{aligned}
&t_{Y,j_1,j_2}\\
&= r_i^2(T^i + t^i_{j_1} + t^i_{j_2}) + r_i(1-r_i)(T^i + t^i_{j_1})\\
&+ r_i(1-r_i)T^{3-i} + t^{3-i}_{j_1}) + (1-r_i)^2(T^{3-i} + t^{3-i}_{j_1} + t^{3-i}_{j_2})\\
&\leq (r_i^2 + r_i(1-r_i))(T^i + t^i_{j_1} + t^i_{j_2}) + ((1-r_i)^2 + r_i(1-r_i))(T^{3-i} + t^{3-i}_{j_1} + t^{3-i}_{j_2})\\
&= r_i(T^i + t^i_{j_1} + t^i_{j_2}) + (1-r_i)(T^{3-i} + t^{3-i}_{j_1} + t^{3-i}_{j_2})\\
&= t_{Y,j}
\end{aligned}$$

The l-task case is similar with m-task case, with $r_i = \frac{1}{2}$. ∎

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 539-550
www.stacs-conf.org

LAGRANGIAN RELAXATION AND PARTIAL COVER (EXTENDED ABSTRACT)

JULIÁN MESTRE [1]

[1] Max-Planck-Institute für Informatik, Saarbrücken, Germany.
E-mail address: jmestre@mpi-inf.mpg.de

Abstract. Lagrangian relaxation has been used extensively in the design of approximation algorithms. This paper studies its strengths and limitations when applied to Partial Cover.

We show that for Partial Cover in general no algorithm that uses Lagrangian relaxation and a Lagrangian Multiplier Preserving (LMP) α-approximation as a black box can yield an approximation factor better than $\frac{4}{3}\alpha$. This matches the upper bound given by Könemann *et al.* (*ESA 2006*, pages 468–479).

Faced with this limitation we study a specific, yet broad class of covering problems: Partial Totally Balanced Cover. By carefully analyzing the inner workings of the LMP algorithm we are able to give an almost tight characterization of the integrality gap of the standard linear relaxation of the problem. As a consequence we obtain improved approximations for the Partial version of Multicut and Path Hitting on Trees, Rectangle Stabbing, and Set Cover with ρ-Blocks.

1. Introduction

Lagrangian relaxation has been used extensively in the design of approximation algorithms for a variety of problems such as k-MST [12, 7, 11], k-median [21, 5], MST with degree constraints [27] and budgeted MST [31].

In this paper we study the strengths and limitations of Lagrangian relaxation applied to the Partial Cover problem. Let $\mathcal{S}$ be collection of subsets of a universal set U with cost $c : \mathcal{S} \to R_+$ and profit $p : U \to R_+$, and let P be a target coverage parameter. A set $\mathcal{C} \subseteq \mathcal{S}$ is a *partial cover* if the overall profit of elements covered by $\mathcal{C}$ is at least P. The objective is to find a minimum cost partial cover.

The high level idea behind Lagrangian relaxation is as follows. In an IP formulation for Partial Cover, the constraint enforcing that at least P profit is covered is *relaxed*: The constraint is multiplied by a parameter λ and lifted to the objective function. This relaxed IP corresponds, up to a constant factor, to the prize-collecting version of the underlying covering problem in which there is no requirement on how much profit to cover but a penalty

1998 ACM Subject Classification: G.2.1.

Key words and phrases: Lagrangian Relaxation, Partial Cover, Primal-Dual Algorithms.

Supported by NSF Award CCF 0430650, a University of Maryland Dean's Dissertation Fellowship, and partly by an Alexander von Humboldt Fellowship.

of $\lambda p(i)$ must be paid if we leave element $i \in U$ uncovered. An approximation algorithm for the prize-collecting version having the Lagrangian Multiplier Preserving (LMP) property[1] is used to obtain values λ_1 and λ_2 that are close together for which the algorithm produces solutions $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. These solutions are such that $\mathcal{C}_1$ is inexpensive but unfeasible (covering less than P profit), and $\mathcal{C}_2$ is feasible (covering at least P profit) but potentially very expensive. Finally, these two solutions are combined to obtain a cover that is both inexpensive and feasible.

Broadly speaking there are two ways to combine $\mathcal{C}_1$ and $\mathcal{C}_2$. One option is to treat the approximation algorithm for the prize-collecting version as a black box, only making use of the LMP property in the analysis. Another option is to focus on a particular LMP algorithm and exploit additional structure that it may offer. Not surprisingly, the latter approach has yielded better approximation guarantees. For example, for k-median compare the 6-approximation of Jain and Vazirani [21] to the 4-approximation of Charikar and Guha [5]; for k-MST compare the 5-factor to the 3-factor approximation due to Garg [12].

The results in this paper support the common belief regarding the inherent weakness of the black-box approach. First, we show a lower bound on the approximation factor achievable for Partial Cover in general using Lagrangian relaxation and the black-box approach that matches the recent upper bound of Könemann et al. [26]. To overcome this obstacle, we concentrate on Kolen's algorithm for Prize-Collecting Totally Balanced Cover [25]. By carefully analyzing the algorithm's inner workings we identify structural similarities between $\mathcal{C}_1$ and $\mathcal{C}_2$, which we later exploit when combining the two solutions. As a result we derive an almost tight characterization of the integrality gap of the standard linear relaxation for Partial Totally Balanced Cover. This in turn implies improved approximation algorithms for a number of related problems.

1.1. Related Work

Much work has been done on covering problems because of both their simple and elegant formulation, and their pervasiveness in different application areas. In its most general form the problem, also known as Set Cover, cannot be approximated within $(1-\epsilon)\ln|U|$ unless $NP \subseteq \mathrm{DTIME}(|U|^{\log\log|U|})$ [9]. Due to this hardness, easier, special cases have been studied.

A general class of covering problems that can be solved efficiently are those whose element-set incidence matrix is balanced. A $0,1$ matrix is *balanced* if it does not contain a square submatrix of odd order with row and column sums equal to 2. These matrices were introduced by Berge [4] who showed that if A is balanced then the polyhedron $\{x \geq 0 : Ax \geq 1\}$ is integral. A $0,1$ matrix is *totally balanced* if it does not contain a square submatrix with row and column sums equal to 2 and no identical columns. Kolen [25] gave a simple primal-dual algorithm that solves optimally the covering problem defined by a totally balanced matrix. A $0,\pm 1$ matrix is *totally unimodular* if every square submatrix has determinant 0 or ± 1. Although totally balanced and totally unimodular matrices are subclasses of balanced matrices, the two classes are neither disjoint nor one is included in the other.

Beyond this point, even minor generalizations can make the covering problem hard. For example, consider the *vertex cover* problem: Given a graph $G = (V, E)$ we are to choose a minimum size subset of vertices such that every edge is incident on at least one of the chosen vertices. If G is bipartite, the element-set incidence matrix for the problem

[1]The definition of the LMP property is outlined in Section 2.

is totally unimodular; however, if G is a general graph the problem becomes NP-hard [24]. Numerous approximation algorithms have been developed for vertex cover [19]. The best known approximation factor for general graphs is $2 - o(1)$ [3, 16, 23]; yet, after 25 years of study, the best constant factor approximation for vertex cover remains 2 [8, 2, 18]. This lack of progress has led researchers to seek generalizations of vertex cover that can still be approximated within twice of optimum. One such generalization is the *multicut* problem on trees: Given a tree T and a collection of pairs of vertices, a cover is formed by a set of edges whose removal separates all pairs. The problem was first studied by Garg et al. [13] who gave an elegant primal-dual 2-approximation.

A notable shortcoming of the standard set cover formulation is that certain hard-to-cover elements, also known as *outliers* [6], can render the optimal solution very expensive. Motivated by the presence of outliers, the unit-profit partial version calls for a collection of sets covering not all but a specified number k of elements. Partial Multicut, a.k.a. k-Multicut, was recently studied independently by Levin and Segev [28] and by Golovin et al. [15], who gave a $\frac{8}{3}+\epsilon$ approximation algorithm. This scheme was generalized by Könemann et al. [26] who showed how to design a $\frac{4}{3}\alpha+\epsilon$ approximation for any covering problem using Lagrangian relaxation and an α-LMP approximation as a black box. (Their algorithm runs in time polynomial on $|U|, |\mathcal{S}|^{\frac{1}{\epsilon}}$ and the running time of the α-LMP approximation.)

1.2. Our Results and Outline of the Paper

Section 3 shows that for Partial Cover in general no algorithm that uses Lagrangian relaxation and an α-LMP approximation as a black box can yield an approximation factor better than $\frac{4}{3}\alpha$. In Section 4 we give an almost tight characterization of the integrality gap of the standard LP for Partial Totally Balanced Cover, settling a question posed by Golovin et al. [15]. Our approach is based on Lagrangian relaxation and Kolen's algorithm. We prove that $\mathrm{IP} \leq \left(1 + \frac{1}{3^{k-1}}\right) \mathrm{LP} + k\, c_{\max}$ for any $k \geq 1$, where IP and LP are the costs of the optimal integral and fractional solutions respectively and $c_{\max}$ is the cost of the most expensive set in the instance. The trade-off between additive and multiplicative error is not an artifact of our analysis or a shortcoming of our approach. On the contrary, this is precisely how the integrality gap behaves. More specifically, we show a family of instances where $\mathrm{IP} > \left(1 + \frac{1}{3^{k-1}}\right) \mathrm{LP} + \frac{k}{2} c_{\max}$. In other words, there is an unbounded additive gap in terms of $c_{\max}$ but as it grows the multiplicative gap narrows exponentially fast.

Finally, we show how the above result can be applied, borrowing ideas from [14, 17, 15], to get a $\rho+\epsilon$ approximation or a quasi-polynomial time ρ-approximation for covering problems that can be expressed with a suitable combination of ρ totally-balanced matrices. This translates into improved approximations for a number of problems: a $2+\epsilon$ approximation for the Partial Multicut on Trees [28, 15], a $4+\epsilon$ approximation for Partial Path Hitting on Trees [30], a 2-approximation for Partial Rectangle Stabbing [14], and a ρ approximation for Partial Set-Cover with ρ-blocks [17]. In addition, the ϵ can be removed from the first two approximation guarantees if we allow quasi-polynomial time. It is worth noting that prior to this work, the best approximation ratio for all these problems could be achieved with the framework of Könemann et al. [26]. In each case our results improve the approximation ratio by a $\frac{4}{3}$ multiplicative factor. Due to lack of space these results only appear in the full version[2] of the paper.

[2]Full version available at `http://arxiv.org/abs/0712.3936`

2. Lagrangian relaxation

Let $\mathcal{S} = \{1, \ldots, m\}$ be a collection of subsets of a universal set $U = \{1, \ldots, n\}$. Each set has a cost specified by $c \in R_+^m$, and each element has a profit specified by $p \in R_+^n$. Given a target coverage P, the objective of the Partial Cover problem is to find a minimum cost solution $\mathcal{C} \subseteq \mathcal{S}$ such that $p(\mathcal{C}) \geq P$, where the notation $p(\mathcal{C})$ denotes the overall profit of elements covered by $\mathcal{C}$. The problem is captured by the IP below. Matrix $A = \{a_{ij}\} \in \{0,1\}^{n \times m}$ is an element-set incidence matrix, that is, $a_{ij} = 1$ if and only if element $i \in U$ belongs to set $j \in \mathcal{S}$; variable x_j indicates whether set j is chosen in the solution $\mathcal{C}$; variable r_i indicates whether element i is left uncovered.

Lagrangian relaxation is used to get rid of the constraint bounding the profit of uncovered elements to be at most $p(U) - P$. The constraint is multiplied by the parameter λ, called Lagrange Multiplier, and is lifted to the objective function. The resulting IP corresponds, up to the constant $\lambda\,(p(U) - P)$ factor in the objective function, to the prize-collecting version of the covering problem, where the penalty for leaving element i uncovered is λp_i.

$$\begin{aligned} &\min\ c \cdot x \\ &Ax + Ir \geq 1 \\ &\quad p \cdot r \leq p(U) - P \\ &\quad r_i, x_j \in \{0,1\} \end{aligned} \qquad \xrightarrow[\text{Relaxation}]{\text{Lagrangian}} \qquad \begin{aligned} &\min\ c \cdot x + \lambda p \cdot r - \lambda\,(p(U) - P) \\ &\quad Ax + Ir \geq 1 \\ &\quad r_i, x_j \in \{0,1\} \end{aligned}$$

Let OPT be the cost of an optimal partial cover and OPT-PC(λ) be the cost of an optimal prize-collecting cover for a given λ. Let $\mathcal{A}$ be an α-approximation for the prize-collecting variant of the problem. Algorithm $\mathcal{A}$ is said to have the Lagrangian Multiplier Preserving (LMP) property if it produces a solution $\mathcal{C}$ such that

$$c(\mathcal{C}) + \alpha\,\lambda\big(p(U) - p(\mathcal{C})\big) \leq \alpha\,\text{OPT-PC}(\lambda). \tag{2.1}$$

Note that OPT-PC(λ) $\leq$ OPT $+ \lambda\,(p(U) - P)$. Thus,

$$c(\mathcal{C}) \leq \alpha\Big(\text{OPT} + \lambda\,\big(p(\mathcal{C}) - P\big)\Big). \tag{2.2}$$

Therefore, if we could find a value of λ such that $\mathcal{C}$ covers exactly P profit then $\mathcal{C}$ is α-approximate. However, if $p(\mathcal{C}) < P$, the solution is not feasible, and if $p(\mathcal{C}) > P$, equation (2.2) does not offer any guarantee on the cost of $\mathcal{C}$. Unfortunately, there are cases where no value of λ produces a solution covering exactly P profit. Thus, the idea is to use binary search to find two values λ_1 and λ_2 that are close together and are such that $\mathcal{A}(\lambda_1)$ covers less, and $\mathcal{A}(\lambda_2)$ covers more than P profit. The two solutions are then combined in some fashion to produce a feasible cover.

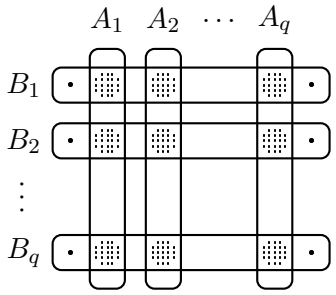
3. Limitations of the black-box approach

A common way to combine the two solutions returned by the α-LMP is to treat the algorithm as a black box, solely relaying on the LMP property (2.1) in the analysis. More formally, an algorithm for Partial Cover that uses Lagrangian relaxation and an α-LMP approximation $\mathcal{A}$ as a black box is as follows. First, we are allowed to run $\mathcal{A}$ with as many different values of λ as desired; then, the solutions thus found are combined to produce a

feasible partial cover. No computational restriction is placed on the second step, except that only sets returned by $\mathcal{A}$ may be used.

Theorem 3.1. *In general, the Partial Cover problem cannot be approximated better than $\frac{4}{3}\alpha$ using Lagrangian relaxation and an α-LMP algorithm $\mathcal{A}$ as a black box.*

Let $A_1, \ldots A_q$ and $B_1, \ldots B_q$ be sets as depicted on the right. For each i and j the intersection $A_i \cap B_j$ consists of a cluster of q elements. There are q^2 clusters. Set A_i is made up of q clusters; set B_i is made up of q clusters and two additional elements (the leftmost and rightmost elements in the picture.) Thus $|A_i| = q^2$ and $|B_i| = q^2 + 2$. In addition, there are sets $O_1, \ldots, O_q$, which are not shown in the picture. Set O_i contains one element from each cluster and the leftmost element of B_i. Thus $|O_i| = q^2 + 1$. The cost of O_i is $\frac{1}{q}$, the cost of A_i is $\frac{2\alpha}{3q}$, and the cost of B_i is $\frac{4\alpha}{3q}$. Every element has unit profit and the target coverage is $P = q^3 + q$. It is not hard to see that $O_1, \ldots, O_q$ is an optimal partial cover with a cost of 1.



The α-LMP approximation algorithm we use has the unfortunate property that it never returns sets from the optimal solution.

Lemma 3.2. *There exists an α-LMP approximation $\mathcal{A}$ that for the above instance and any value of λ outputs either $\emptyset$ or $A_1, \ldots, A_q$ or $B_1, \ldots, B_q$.*

The proof that such an algorithm exists is given in the full version of the paper. Hence, if we use $\mathcal{A}$ as a black box we must build a partial cover with the sets $A_1, \ldots, A_q$ and $B_1, \ldots, B_q$. Note that in order to cover $q^2 + q$ elements either all A-sets, or all B-sets must be used. In the first case $\frac{q}{2}$ additional B-sets are needed to attain feasibility, and the solution has cost $\frac{4}{3}\alpha$; in the second case the solution is feasible but again has cost $\frac{4}{3}\alpha$. Theorem 3.1 follows.

One assumption usually made in the literature [1, 10, 26] is that $c_{\max} = \max_j c_j \leq \epsilon\,\mathrm{OPT}$, for some constant $\epsilon > 0$, or more generally an additive error in terms of $c_{\max}$ is allowed. This does not help in our construction as $c_{\max}$ can be made arbitrarily small by increasing q.

Admittedly, our lower bound example belongs to a specific class of covering problem (every element belongs to at most three sets) and although the example can be embedded into a partial totally unimodular covering problem (see full version), it is not clear how to embed the example into other classes. Nevertheless, the $\frac{4}{3}\alpha$ upper bound of Koneman et el. [26] makes no assumption about the underlying problem, only using the LMP property (2.1) in the analysis. It was entirely conceivable that the $\frac{4}{3}\alpha$ factor could be improved using a different merging strategy—Theorem 3.1 precludes this possibility.

4. Partial Totally Balanced Cover

In order to overcome the lower bound of Theorem 3.1, one must concentrate on a specific class of covering problems or make additional assumptions about the α-LMP algorithm. In this section we focus on covering problems whose IP matrix A is totally balanced. More specifically, we study the integrality gap of the standard linear relaxation for Partial Totally Balanced Cover (P-TBC) shown below.

Theorem 4.1. *Let* IP *and* LP *be the cost of the optimal integral and fractional solutions of an instance of P-TBC. Then* $\text{IP} \leq \left(1 + \frac{1}{3^{k-1}}\right) \text{LP} + k\, c_{\max}$ *for any* $k \in Z_+$. *Furthermore, for any large enough* $k \in Z_+$ *the exists an instance where* $\text{IP} > \left(1 + \frac{1}{3^{k-1}}\right) \text{LP} + \frac{k}{2}\, c_{\max}$.

$$\begin{array}{ll} \min\ c \cdot x & \max\ 1 \cdot y - (p(U) - P)\,\lambda \\ Ax + Ir \geq 1 & A^T y \leq c \\ p \cdot r \leq p(U) - P & y \leq \lambda p \\ r_i, x_e \geq 0 & y_i, \lambda \geq 0 \end{array} \qquad \xrightarrow[\text{LP Duality}]{}$$

The rest of this section is devoted to proving the upper bound in Theorem 4.1, the lower bound is left for the full version of the paper. Our approach is based on Lagrangian relaxation and Kolen's algorithm for Prize-Collecting Totally Balanced Cover (PC-TBC). The latter exploits the fact that a totally balanced matrix can be put into greedy standard form by permuting the order of its rows and columns; in fact, the converse is also true [20]. A matrix is in standard greedy form if it does not contain as an induced submatrix

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \tag{4.1}$$

There are polynomial time algorithms that can transform a totally balanced matrix into greedy standard form [32] by shuffling the rows and columns of A. Since this transformation does not affect the underlying covering problem, we assume that A is given in standard greedy form.

4.1. Kolen's algorithm for Prize-Collecting Totally Balanced Cover

For the sake of completeness we describe Kolen's primal-dual algorithm for PC-TBC. The algorithm finds a dual solution y and a primal solution C, which is then pruned in a reverse-delete step to obtain the final solution $\widehat{C}$. The linear and dual relaxations for PC-TBC appear below.

$$\begin{array}{ll} \min\ c \cdot x + \lambda p \cdot r & \max\ 1 \cdot y \\ Ax + Ir \geq 1 & A^T y \leq c \\ r_i, x_e \geq 0 & y \leq \lambda p \\ & y_i \geq 0 \end{array} \qquad \xrightarrow[\text{LP Duality}]{}$$

The residual cost of the set j w.r.t. y is defined as $c'_j = c_j - \sum_{i:a_{ij}=1} y_i$. The algorithm starts from the trivial dual solution $y = 0$, and processes the elements in increasing column order of A^T. Let i the index of the current element. Its corresponding dual variable, y_i, is increased until either the residual cost of some set j containing i equals 0 (we say set j becomes tight), or y_i equals λp_i (Lines 3-5).

Let $C = \{j \,|\, c'_j = 0\}$ be the set of tight sets after the dual update is completed. As it stands the cover C may be too expensive to be accounted for using the lower bound provided by $1 \cdot y$ because a single element may belong to multiple sets in C. The key insight is that some of the sets in C are redundant and can be pruned.

```
KOLEN((A, c, p, λ))                          ()
 1  // Dual update                           7  // Reverse delete
 2  y ← 0, C ← ∅, Ĉ ← ∅                     8  while C ≠ ∅
 3  for i ← 1 to n                          9  do j ← largest set index in C
 4  do δ ← min{c'_j | a_ij = 1}            10      Ĉ ← Ĉ + j
 5       y_i ← min{λp_i, δ}                11      C ← C \ { j' | j dominates j' or j = j' }
 6  C ← {j | c'_j = 0}                     12  return (Ĉ, y)
```

Definition 4.2. Given sets j_1, j_2 we say that j_1 *dominates* j_2 in y if $j_1 > j_2$ and there exists an item i such that $y_i > 0$ and i belongs to j_1 and j_2, that is, $a_{ij_1} = a_{ij_2} = 1$.

The reverse-delete step iteratively identifies the largest index j in C, adds j to $\widehat{C}$, and removes j and all the sets it dominates. This is repeated until no set is left in C (Lines 8–11).

Notice that all sets $j \in C$ are tight, thus we can pay for set j by charging the dual variables of items that belong to j. Because of the reverse-delete step if $y_i > 0$ then i belongs to at most one set in $\widehat{C}$; thus in paying for $\widehat{C}$ we charge covered items at most once. Using the fact A is in standard greedy form, it can be shown [25] that if i was left uncovered then we can afford its penalty, i.e., $y_i = \lambda p_i$. The solution $\widehat{C}$ is optimal for PC-TBC since

$$\sum_{j \in \widehat{C}} c_j \;+\; \sum_{\substack{i \in U \text{ s.t.} \\ \nexists\, j \in \widehat{C} : a_{ij} = 1}} \lambda p_i \;=\; \sum_{\substack{i \in U \text{ s.t.} \\ \exists\, j \in \widehat{C} : a_{ij} = 1}} y_i + \sum_{\substack{i \in U \text{ s.t.} \\ \nexists\, j \in \widehat{C} : a_{ij} = 1}} y_i \;=\; \sum_{i \in U} y_i. \tag{4.2}$$

If we could find a value of λ such that KOLEN(A, c, p, λ) returns a solution $(\widehat{C}, y)$ covering *exactly* P profit, we are done since from (4.2) it follows that

$$\sum_{j \in \widehat{C}} c_j = \sum_{i \in U} y_i - \lambda\,(p(U) - P). \tag{4.3}$$

Notice that (y, λ) is a feasible for the dual relaxation of P-TBC and its cost is precisely the right hand side of (4.3). Therefore for this instance IP=DL=LP and Theorem 4.1 follows.

Unfortunately, there are cases where no such value of λ exists. Nonetheless, we can always find a *threshold value* λ such that for any infinitesimally small $\delta > 0$, $\lambda^- = \lambda - \delta$ and $\lambda^+ = \lambda + \delta$ produce solutions covering less and more than P profit respectively. A threshold value can be found using Megiddo's parametric search [29] by making $O(n \log m)$ calls to the procedure KOLEN.

Let y (y^-) be the dual solution and C (C^-) the set of tight sets when KOLEN is run on λ (λ^-). Without loss of generality assume $\widehat{C}$ covers more than P profit. (The case where $\widehat{C}$ covers less than P profit is symmetrical: we work with y^+ and C^+ instead of y^- and C^-.)

Our plan to prove Theorem 4.1 is to devise an algorithm to merge $\widehat{C}$ and $\widehat{C}^-$ in order to obtain a cheap solution covering at least P profit.

4.2. Merging two solutions

Before describing the algorithm we need to establish some important properties regarding these two solutions and their corresponding dual solutions.

For any i, the value of y_i^- is a linear function of δ for all i. This follows from the fact that δ is infinitesimally small. Furthermore, the constant term in this linear function is y_i.

Lemma 4.3. *For each $i \in U$ there exists $a \in Z$, independent of δ, such that $y_i^- = y_i + a\delta$.*

Proof. By induction on the number of iteration of the dual update step of KOLEN, using the fact that the same property holds for the residual cost of the sets. ∎

A useful corollary of Lemma 4.3 is that $C^- \subseteq C$, since if the residual cost of a set is non-zero in y it must necessarily be non-zero in y^-. The other way around may not hold.

At the heart of our approach is the notion of a merger graph $G = (V, E)$. The vertex set of G is made up of sets from the two solutions, i.e., $V = \widehat{C} \oplus \widehat{C}^-$. The edges of G are directed and given by

$$E = \left\{ (j_1, j_2) \;\middle|\; \begin{array}{l} j_1 \in \widehat{C}^- \setminus \widehat{C},\ j_2 \in \widehat{C} \setminus \widehat{C}^- \text{ s.t. } j_1 \text{ dominates } j_2 \text{ in } y^-, \text{ or} \\ j_1 \in \widehat{C} \setminus \widehat{C}^-,\ j_2 \in \widehat{C}^- \setminus \widehat{C} \text{ s.t. } j_1 \text{ dominates } j_2 \text{ in } y \end{array} \right\} \tag{4.4}$$

This graph has a lot of structure that can be exploited when merging the solutions.

Lemma 4.4. *The merger graph $G = (V, E)$ of $\widehat{C}^-$ and $\widehat{C}$ is a forest of out-branchings.*

Proof. First note that G is acyclic, since if $(j_1, j_2) \in E$ then necessarily $j_1 > j_2$. Thus, it is enough to show that the in-degree of every $j \in V$ is at most one. Suppose otherwise, that is, there exist $j_1, j_2 \in V$ such that $(j_1, j), (j_2, j) \in E$. Assume that $j_1 < j_2$ and $j \in \widehat{C}$ (the remaining cases are symmetrical).

By definition (4.4), we know that $j_1\,(j_2) \in \widehat{C}^-$ and that there exists i_1 (i_2) that belongs to j and j_1 (j_2) such that $y^-_{i_1} > 0$ ($y^-_{i_2} > 0$). Since A^T is in standard greedy form we can infer that i_2 belongs to j_1 if $i_1 < i_2$, or i_1 belongs to j_2 if $i_1 > i_2$: The diagram on the right shows how, using the fact that A^T does not contain (4.1) as an induced submatrix, we can infer that the boxed entries must be 1. In either case we get that j_2 dominates j_1 in y^-, which contradicts the fact that both belong to $\widehat{C}^-$. ∎

	i_1	i_2	i_2	i_1
j	1	1	1	1
j_1	1	[1]		1
j_2		1	1	[1]

```
MERGE((Ĉ⁻, Ĉ))
1  let G be the merger graph for Ĉ⁻ and Ĉ
2  D ← Ĉ⁻
3  for each root r in G
4  do if p(D ⊕ T_r) ≤ P
5        then then D ← D ⊕ T_r
6             else return INCREASE(r, D)
```

The procedure MERGE starts from the unfeasible solution $D = \widehat{C}^-$ and guided by the merger graph G, it modifies D step by step until feasibility is attained. The operation used to update D is to take the symmetric difference of D and a subtree of G rooted at a vertex $r \in V$, which we denote by T_r. For each root r of an out-branchings of G we set $D \leftarrow D \oplus T_r$, until $p(D \oplus T_r) > P$. At this point we return the solution produced by INCREASE(r, D).

Notice that after setting $D \leftarrow D \oplus T_r$ in Line 5, the solution D "looks like" $\widehat{C}$ within T_r. Indeed, if all roots are processed then $D = \widehat{C}$. Therefore, at some point we are bound

to have $p(D \oplus T_r) > P$ and to make the call INCREASE(r, D) in Line 6. Before describing INCREASE we need to define a few terms. Let the *absolute benefit* of set j, which we denote by b_j, be the profit of elements uniquely covered by set j, that is,

$$b_j = p\left(\left\{\ i \in U\ \mid\ \forall\, j' \in \widehat{C} \cup \widehat{C}^{-} : a_{ij'} = 1 \text{ iff } j' = j\ \right\}\right). \tag{4.5}$$

Let $D \subseteq \widehat{C} \cup \widehat{C}^{-}$. Note that if $j \in D$, the removal of j decreases the profit covered by D by at least b_j; on the other hand, if $j \notin D$, its addition increases the profit covered by at least b_j. This notion of benefit can be extended to subtrees,

$$\Delta(T_j, D) = \sum_{j' \in T_j \setminus D} b_{j'} - \sum_{j' \in T_j \cap D} b_{j'}. \tag{4.6}$$

We call this quantity the *relative benefit* of T_j with respect to D. It shows how the profit of uniquely covered elements changes when we take $D \oplus T_j$. Note that $\Delta(T_j, D)$ can positive or negative.

Everything is in place to explain INCREASE(j, D). The algorithm assumes the input solution is unfeasible but can be made feasible by adding some sets in T_j; more precisely, we assume $p(D) \leq P$ and $P < p(D) + \Delta(T_j, D)$. If adding j to D makes the solution feasible then return $D + j$ (Lines 2-3). If there exists a child c of j that can be used to propagate the call down the tree then do that (Lines 4-5). Otherwise, *split* the subtree T_j: Add j to D and process the children of c, setting $D \leftarrow D \oplus T_c$ until D becomes feasible (Lines 6-9). At this point $p(D) > P$ and $p(D \oplus T_c) \leq P$. If $P - p(D \oplus T_c) < p(D) - P$ then call INCREASE$(c, D \oplus T_c)$ else call DECREASE(c, D) and let D' be the cover returned by the recursive call (Lines 10-12). Finally, return the cover with minimum cost between D and D'.

```
INCREASE((j, D))
 1  // assume p(D) ≤ P < p(D) + Δ(T_j, D)
 2  if p(D + j) ≥ P
 3     then return D + j
 4  if ∃ child c of j : p(D) + Δ(T_c, D) > P
 5     then return INCREASE(c, D)
 6  D ← D + j
 7  while p(D) ≤ P
 8  do c ← child of j maximizing Δ(T_c, D)
 9       D ← D ⊕ T_c
10  if P − p(D ⊕ T_c) < p(D) − P
11     then D' ← INCREASE(c, D ⊕ T_c)
12     else  D' ← DECREASE(c, D)
13  return min cost {D, D'}
```

```
DECREASE((j, D))
 1  // assume p(D) ≥ P > p(D) + Δ(T_j, D)
 2  if p((D ⊕ T_j) + j) ≥ P
 3     then return (D ⊕ T_j) + j
 4  if ∃ child c of j : p(D) + Δ(T_c, D) < P
 5     then return DECREASE(c, D)
 6  D ← D + j
 7  while p(D) ≥ P
 8  do c ← child of j minimizing Δ(T_c, D)
 9       D ← D ⊕ T_c
10  if p(D ⊕ T_c) − P < P − p(D)
11     then D' ← INCREASE(c, D)
12     else  D' ← DECREASE(c, D ⊕ T_c)
13  return min cost {D ⊕ T_c, D'}
```

The twin procedure DECREASE(j, D) is essentially symmetrical: Initially the input is feasible but can be made unfeasible by removing some sets in T_j; more precisely $p(D) \geq P$ and $P < p(D) + \Delta(T_c, D)$.

At a very high level, the intuition behind the INCREASE/DECREASE scheme is as follows. In each call one of three things must occur:

(i) A feasible cover with a small coverage excess is found (Lines 2-3), or
(ii) The call is propagated down the tree at no cost (Lines 4-5), or
(iii) A subtree T_j is split (Lines 6-9). In this case, the cost c_j cannot be accounted for, but the offset in coverage $|P - p(D)|$ is reduced at least by a factor of 3.

If the INCREASE/DECREASE algorithms split many subtrees (incurring a high extra cost) then the offset in coverage must have been very high at the beginning, which means the cost of the dual solution is high and so the splitting cost can be charged to it. In order to flesh out these ideas into a formal proof we need to establish some crucial properties of the merger graph and the algorithms. Proofs are omitted due to lack of space.

Lemma 4.5. *If $y_i < \lambda\, p_i$ then there exist $j' \in \widehat{C}$ and $j'' \in \widehat{C}^-$ such that either $j' = j''$ or $(j', j'') \in E$ or $(j'', j') \in E$.*

Lemma 4.6. *Let (j, D) be the input of* INCREASE/DECREASE. *Then at the beginning of each call we have $j' \in D$ or $j'' \in D$ for all $(j', j'') \in E$. Furthermore, if $j' \in D$ and $j'' \in D$ then j' or j'' must have been split in a previous call.*

Lemma 4.7. *Let (j, D) be the input of* INCREASE/DECREASE. *For* INCREASE *we always have $p(D) \le P < p(D) + \Delta(T_j, D)$, and for* DECREASE *we have $p(D) \ge P > p(D) + \Delta(T_j, D)$.*

Recall that y is also a feasible solution for the dual relaxation of P-TBC and its cost is given by $\mathrm{DL} = \sum_{i=1}^{n} y_i - (p(U) - P)\lambda$. The following lemma proves the upper bound of Theorem 4.1.

Lemma 4.8. *Suppose* MERGE *outputs D. Then $c(D) \le \big(1 + \frac{1}{3^{k-1}}\big)\mathrm{DL} + k\, c_{\max}$ for all $k \in Z_+$.*

Proof. Let us digress for a moment for the sake of exposition. Suppose that in Line 6 of MERGE, instead of calling INCREASE, we return $D' = D \oplus T_r$. Notice every arc in the merger graph has exactly one endpoint in D'. By Lemma 4.5, any element i not covered by D' must have $y_i = \lambda\, p_i$. Furthermore, if $y_i > 0$ then there exists at most one set in D' that covers i; if two such sets exist, one must dominate the other in y and y^-, which is not possible. Hence,

$$c(D) = \sum_{j \in D'} \sum_{i : a_{ij} = 1} y_i = \sum_{\substack{i \text{ s.t.} \\ \exists\, j \in D' : a_{ij} = 1}} y_i = \sum_{i \in U} y_i - (p(U) - p(D'))\lambda \le \mathrm{DL} + (p(D') - P)\lambda \quad (4.7)$$

In the fortunate case that $(p(D') - P)\lambda \le k c_{\max}$, the lemma would follow. Of course, this need not happen and this is why we make the call to INCREASE instead of returning D'.

Let j_q be the root of the q^{th} subtree split by INCREASE/DECREASE. Also let D_q the solution right before splitting T_{j_q}, and D'_q and D''_q be the unfeasible/feasible pair of solutions after the splitting, which are used as parameters in the recursive calls (Lines 11-12). Suppose Lines 7-9 processed only one child of j_q, this can only happen in INCREASE, in which case $p(D''_q) > P$ but $p(D''_q) - b_{j_q} < P$. The same argument used to derive (4.7) gives us

$$c\left(D''_q \setminus \{j_{\le q}\}\right) \le \sum_{i \in U} y_i - \left(p(U) - p(D''_q) + b_{j_q}\right)\lambda \le \mathrm{DL} \quad (4.8)$$

The cost of the missing sets is $c(\{j_{\le q}\}) \le q\, c_{\max}$, thus if $q \le k$ the lemma follows. A similar bound can be derived if the recursive call ends in Line 3 before splitting the k^{th} subtree.

Finally, the last case to consider is when Lines 7-9 process two or more children j_q for all $q \leq k$. In this case

$$|p(D_q) - P| \geq 3 \min\left\{|p(D'_q) - P|, |p(D''_q) - P|\right\} = 3\,|p(D_{q+1}) - P|, \tag{4.9}$$

which implies $|p(D_1) - P| \geq 3^{k-1}|p(D_k) - P| \geq 3^{k-1}|p(D''_k) - P|$. Also, $\lambda(P - p(D_1)) \leq \mathrm{DL}$ since all elements i not covered by D_1 must be such that $y_i = \lambda p_i$. Hence, as before

$$c\left(D''_k \setminus \{j_{\leq k}\}\right) \leq \mathrm{DL} + \left(p(D''_k) - P\right)\lambda \leq \mathrm{DL} + \frac{P - p(D_1)}{3^{k-1}} \leq \left(1 + \frac{1}{3^{k-1}}\right)\mathrm{DL} \tag{4.10}$$

Adding the cost of $\{j_{\leq k}\}$ we get the lemma. ∎

5. Concluding remarks and open problems

The results in this paper suggest that Lagrangian relaxation is a powerful technique for designing approximation algorithms for partial covering problems, even though the black-box approach may not be able to fully realize its potential.

It would be interesting to extend this study on the strengths and limitation of Lagrangian relaxation to other problems. The obvious candidate is the k-Median problem. Jain and Vazirani [21] designed a 2α-approximation for k-Median using as a black box an α-LMP approximation for Facility Location. Later, Jain et al. [22] gave a 2-LMP approximation for Facility Location. Is the algorithm in [21] optimal in the sense of Theorem 3.1? Can the algorithm in [22] be turned into a 2-approximation for k-Median by exploiting structural similarities when combining the two solutions?

Acknowledgments: I am indebted to Danny Segev for sharing an early draft of [26] and for pointing out Kolen's work. Also thanks to Mohit Singh and Arie Tamir for helpful discussions and to Elena Zotenko for suggesting deriving the result of Section 3.

References

[1] S. Arora and G. Karakostas. A $2+\epsilon$ approximation algorithm for the k-MST problem. In *Proc., 11th Symposium on Discrete Algorithms*, pp. 754–759, 2000.

[2] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for approximating the weighted vertex cover. *Journal of Algorithms*, 2:198–203, 1981.

[3] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.

[4] C. Berge. Balanced matrices. *Mathematical Programming*, 2:19–31, 1972.

[5] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proc., Symp. on Foundations of Computer Science*, pp. 378–388, 1999.

[6] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *Proc., 12th Symposium on Discrete Algorithms*, pp. 642–651, 2001.

[7] F. A. Chudak, T. Roughgarden, and D. P. Williamson. Approximate k-MSTs and k-Steiner trees via the primal-dual method and Lagrangean relaxation. In *Proceedings of the 9th Integer Programming and Combinatorial Optimization Conference (IPCO)*, pp. 60–70, 2001.

[8] K. L. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16(1):23–25, 1983.

[9] U. Feige. A threshold of ln n for approximating set cover. *J. of the ACM*, 45(4):634–652, 1998.

[10] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

[11] N. Garg. Saving an epsilon: a 2-approximation for the k-MST problem in graphs. In *Proc., 37th Symposium on Theory of computing*, pp. 396–402, 2005.
[12] N. Garg. A 3-approximation for the minimum tree spanning k vertices. In *Proc., 37hd Symposium on Foundations of Computer Science*, pp. 302–309, 1996.
[13] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.
[14] D. R. Gaur, T. Ibaraki, and R. Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. *Journal of Algorithms*, 43 (1):138–152, 2002.
[15] D. Golovin, V. Nagarajan, and M. Sing. Approximating the k-multicut problem. In *7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006.
[16] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, 2002.
[17] R. Hassin and D. Segev. Rounding to an integral program. In *Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA'05)*, pp. 44–54, 2005.
[18] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.
[19] D. S. Hochbaum, editor. *Approximation Algorithms for NP–hard Problems*. PWS Publishing Company, 1997.
[20] A. J. Hoffman, A. Kolen, and M. Sakarovitch. Totally-balanced and greedy matrices. *SIAM Journal on Algebraic and Discrete Methods*, 6:721–730, 1985.
[21] K. Jain and V. V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2): 274–296, 2001.
[22] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6): 795–824, 2003.
[23] G. Karakostas. A better approximation ratio for the vertex cover problem. In *Proc., 15th International Colloquium on Automata, Languages, and Programming*, pp. 1043–1050, 2005.
[24] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pp. 85–103. Plenum Press, 1972.
[25] A. Kolen. *Location problems on trees and in the rectilinear plane*. PhD thesis, Matematisch Centrum, Amsterdam, 1982.
[26] J. Könemann, O. Parekh, and D. Segev. A unified approach to approximating partial covering problems. In *Proc. 14th European Symposium on Algorithms (ESA)*, pp. 468–479, 2006.
[27] J. Könemman and R. Ravi. A matter of degree: Improved approximation algorithms for degree-bounded minimum spanning trees. *SIAM Journal of Computing*, 31:1783–1793, 2002.
[28] A. Levin and D. Segev. Partial multicuts in trees. In *Proc. 3rd Intern. Workshop on Approx. and Online Algorithms (WAOA)*, 2005.
[29] N. Megiddo. Combinatorial optimization with rational objective functions. In *Proceedings of the tenth annual ACM symposium on Theory of computing (STOC)*, pp. 1–12, 1978.
[30] O. Parekh and D. Segev. Path hitting in acyclic graphs. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pp. 564–575, 2006.
[31] R. Ravi and M. X. Goemans. The constrained minimum spanning tree problem (extended abstract). In *5th Scandinavian Workshop on Algorithm Theory (SWAT)*, pp. 66–75, 1996.
[32] J. P. Spinard. Doubly lexical ordering of dense 0-1 matrices. *Information Processing Letters*, 45:229–235, 1993.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 551-560
www.stacs-conf.org

ON DYNAMIC BREADTH-FIRST SEARCH IN EXTERNAL-MEMORY

ULRICH MEYER [1]

[1] Institute for Computer Science, J. W. Goethe University, 60325 Frankfurt/Main, Germany
E-mail address: umeyer@ae.cs.uni-frankfurt.de

Abstract. We provide the first non-trivial result on dynamic breadth-first search (BFS) in external-memory: For general sparse undirected graphs of initially n nodes and $O(n)$ edges and monotone update sequences of either $\Theta(n)$ edge insertions or $\Theta(n)$ edge deletions, we prove an amortized high-probability bound of $O(n/B^{2/3} + \mathrm{sort}(n) \cdot \log B)$ I/Os per update. In contrast, the currently best approach for static BFS on sparse undirected graphs requires $\Omega(n/B^{1/2} + \mathrm{sort}(n))$ I/Os.

1. Introduction

Breadth first search (BFS) is a fundamental graph traversal strategy. It can also be viewed as computing single source shortest paths on unweighted graphs. It decomposes the input graph $G = (V, E)$ of n nodes and m edges into at most n levels where level i comprises all nodes that can be reached from a designated source s via a path of i edges, but cannot be reached using less than i edges.

The objective of a dynamic graph algorithm is to efficiently process an online sequence of update and query operations; see [8, 14] for overviews of classic and recent results. In our case we consider BFS under a sequence of either $\Theta(n)$ edge insertions, but not deletions (*incremental* version) or $\Theta(n)$ edge deletions, but not insertions (*decremental* version). After each edge insertion/deletion the updated BFS level decomposition has to be output.

1.1. Computation models.

We consider the commonly accepted external-memory (EM) model of Aggarwal and Vitter [1]. It assumes a two level memory hierarchy with faster internal memory having a capacity to store M vertices/edges. In an I/O operation, one block of data, which can store B vertices/edges, is transferred between disk and internal memory. The measure of performance of an algorithm is the number of I/Os it performs. The number of I/Os needed to read N contiguous items from disk is $\mathrm{scan}(N) = \Theta(N/B)$. The number of I/Os required

1998 ACM Subject Classification: F.2.2.
Key words and phrases: External Memory, Dynamic Graph Algorithms, BFS, Randomization.
Partially supported by the DFG grant ME 3250/1-1, and by the center of massive data algorithmics (MADALGO) funded by the Danish National Research Foundation.

to sort N items is $\mathrm{sort}(N) = \Theta((N/B)\log_{M/B}(N/B))$. For all realistic values of N, B, and M, $\mathrm{scan}(N) < \mathrm{sort}(N) \ll N$.

There has been a significant number of publications on external-memory graph algorithms; see [12, 16] for recent overviews. However, we are not aware of any dynamic graph algorithm in the fully external-memory case (where $|V| > M$).

1.2. Results.

We provide the first non-trivial result on dynamic BFS in external-memory. For general sparse undirected graphs of initially n nodes and $O(n)$ edges and either $\Theta(n)$ edge insertions or $\Theta(n)$ edge deletions, we prove an amortized high-probability bound of $O(n/B^{2/3}+\mathrm{sort}(n)\cdot \log B)$ I/Os per update. In contrast, the currently best bound for static BFS on sparse undirected graphs is $O(n/B^{1/2} + \mathrm{sort}(n))$ I/Os [11].

Also note that for general sparse graphs and worst-case monotone sequences of $\Theta(n)$ updates in *internal-memory* there is asymptotically no better solution than performing $\Theta(n)$ runs of the linear-time static BFS algorithm, even if after each update we are just required to report the changes in the BFS tree (see Fig. 1 for an example). In case $\Omega(n/B^{1/2} + \mathrm{sort}(n))$ I/Os should prove to be a lower bound for static BFS in external-memory, then our result yields an interesting differentiator between static vs. dynamic BFS in internal and external memory.

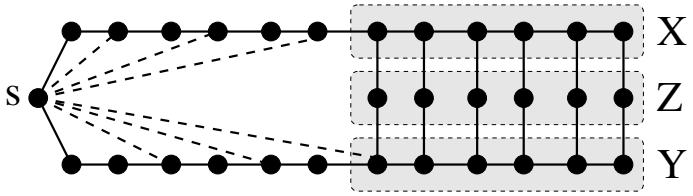


Figure 1: Example for a graph class where each update requires $\Omega(n)$ changes in the BFS tree: inserting new (dashed) edges alternatingly shortcut the distances from s to **X** and s to **Y**. As a result, in the updated BFS tree the parents of all vertices in **Z** keep on changing between **X** and **Y**.

1.3. Organization of the paper.

In Section 2 we will review known BFS algorithms for static undirected graphs. Then we consider traditional and new external-memory methods for graph clustering (Section 3). Subsequently, in Section 4 we provide the new algorithm and analyze it in Section 5. Final remarks concerning extensions and open problems are given in Sections 6 and 7, respectively.

2. Review of Static BFS Algorithms

Internal-Memory. BFS is well-understood in the RAM model. There exists a simple linear time algorithm [6] (hereafter referred as IM_BFS) for the BFS traversal in a graph. IM_BFS keeps a set of appropriate candidate nodes for the next vertex to be visited in a FIFO queue Q. Furthermore, in order to find out the unvisited neighbors of a node from its adjacency list, it marks the nodes as either visited or unvisited.

Unfortunately, as the storage requirements of the graph starts approaching the size of the internal memory, the running time of this algorithm deviates significantly from the predicted $O(n+m)$ asymptotic performance of the RAM model: checking whether edges lead to already visited nodes altogether needs $\Theta(m)$ I/Os in the worst case; unstructured indexed access to adjacency lists may add another $\Theta(n+m/B)$ I/Os.

EM-BFS for dense undirected graphs. The algorithm by Munagala and Ranade [13] (referred as MR_BFS) ignores the second problem but addresses the first by exploiting the fact that the neighbors of a node in BFS level $t-1$ are all in BFS levels $t-2$, $t-1$ or t. Let $L(t)$ denote the set of nodes in BFS level t, and let $A(t)$ be the multi-set of neighbors of nodes in $L(t-1)$. Given $L(t-1)$ and $L(t-2)$, MR_BFS builds $L(t)$ as follows: Firstly, $A(t)$ is created by $|L(t-1)|$ random accesses to get hold of the adjacency lists of all nodes in $L(t-1)$. Thereafter, duplicates are removed from $A(t)$ to get a sorted set $A'(t)$. This is done by sorting $A(t)$ according to node indices, followed by a scan and compaction phase. The set $L(t) := A'(t) \setminus \{L(t-1) \cup L(t-2)\}$ is computed by scanning "in parallel" the sorted sets of $A'(t), L(t-1)$, and $L(t-2)$ to filter out the nodes already present in $L(t-1)$ or $L(t-2)$. The resulting worst-case I/O-bound is $O\left(\sum_t L(t) + \sum_t \mathrm{sort}(A(t))\right) = O\left(n + \mathrm{sort}(n+m)\right)$. The algorithm outputs a BFS-level decomposition of the vertices, which can be easily transformed into a BFS tree using $O(\mathrm{sort}(n+m))$ I/Os [4].

EM-BFS for sparse undirected graphs. Mehlhorn and Meyer suggested another approach [11] (MM_BFS) which involves a preprocessing phase to restructure the adjacency lists of the graph representation. It groups the vertices of the input graph into disjoint clusters of small diameter in G and stores the adjacency lists of the nodes in a cluster contiguously on the disk. Thereafter, an appropriately modified version of MR_BFS is run. MM_BFS exploits the fact that whenever the first node of a cluster is visited then the remaining nodes of this cluster will be reached soon after. By spending only one random access (and possibly, some sequential accesses depending on cluster size) to load the whole cluster and then keeping the cluster data in some efficiently accessible data structure (pool) until it is all processed, on sparse graphs the total amount of I/Os can be reduced by a factor of up to $\sqrt{B}$: the neighboring nodes of a BFS level can be computed simply by scanning the pool and not the whole graph. Though some edges may be scanned more often in the pool, unstructured I/Os to fetch adjacency lists is considerably reduced, thereby reducing the total number of I/Os.

3. Preprocessing

3.1. Traditional preprocessing within MM_BFS.

Mehlhorn and Meyer [11] proposed the algorithms MM_BFS_R and MM_BFS_D, out of which the first is randomized and the second is deterministic. In MM_BFS_R, the partitioning is generated "in parallel rounds": after choosing master nodes independently and uniformly at random, in each round, each master node tries to capture all unvisited neighbors of its current sub-graph into its partition, with ties being resolved arbitrarily.

A similar kind of randomized preprocessing is also applied in parallel [15] and streaming [7] settings. There, however, a dense compressed graph among the master nodes is produced, causing rather high parallel work or large total streaming volume, respectively.

The MM_BFS_D variant first builds a spanning tree T_s for the connected component of G that contains the source node. Arge et al. [2] show an upper bound of $O((1 + \log\log(B \cdot n/m)) \cdot \mathrm{sort}(n + m))$ I/Os for computing such a spanning tree. Each undirected edge of T_s is then replaced by two oppositely directed edges. Note that a bi-directed tree always has at least one Euler tour. In order to construct the Euler tour around this bi-directed tree, each node chooses a cyclic order [3] of its neighbors. The successor of an incoming edge is defined to be the outgoing edge to the next node in the cyclic order. The tour is then broken at the source node and the elements of the resulting list are then stored in consecutive order using an external memory list-ranking algorithm; Chiang et al. [5] showed how to do this in sorting complexity. Thereafter, we chop the Euler tour into *chunks* of $\max\{1, \sqrt{\frac{n \cdot B}{n+m}}\}$ nodes and remove duplicates such that each node only remains in the first chunk it originally occurs; again this requires a couple of sorting steps. The adjacency lists are then re-ordered based on the position of their corresponding nodes in the chopped duplicate-free Euler tour: all adjacency lists for nodes in the same chunks form a cluster and the distance in G between any two vertices whose adjacency-lists belong to the same cluster is bounded by $\max\{1, \sqrt{\frac{n \cdot B}{n+m}}\}$.

3.2. Modified preprocessing for dynamic BFS.

The preprocessing methods for the static BFS in [11] may produce very unbalanced clusters: for example, with MM_BFS_D using chunk size $1 < \mu < O(\sqrt{B})$ there may be $\Omega(n/\mu)$ clusters being in charge of only $O(1)$ adjacency-lists each. For the dynamic version, however, we would like to argue that each random access to a cluster not visited so far provides us with $\Omega(\mu)$ new adjacency-lists. Unfortunately, finding such a clustering I/O-efficiently seems to be quite hard. Therefore, we shall already be satisfied with an Euler tour based randomized construction ensuring that the *expected* number of adjacency-lists kept in all but one[1] clusters is $\Omega(\mu)$.

The preprocessing from MM_BFS_D is modified as follows: each vertex v in the spanning tree T_s is assigned an independent binary random number $r(v)$ with $\mathbf{P}[r(v) = 0] = \mathbf{P}[r(v) = 1] = 1/2$. When removing duplicates from the Euler tour, instead of storing v's adjacency-list in the cluster related to the chunk with the *first* occurrence of a vertex v, now we only stick to its first occurrence iff $r(v) = 0$ and otherwise ($r(v) = 1$) store v's adjacency-list in the cluster that corresponds to the *last* chunk of the Euler tour v appears in. For leaf nodes v, there is only one occurrence on the tour, hence the value of $r(v)$ is irrelevant. Obviously, each adjacency-lists is stored only once. Furthermore, the modified procedure maintains all good properties of the standard preprocessing within MM_BFS_D like guaranteed bounded distances of $O(\mu)$ in G between the vertices belonging to the same cluster and $O(n/\mu)$ clusters overall.

Lemma 3.1. *For chunk size $\mu > 1$ and each but the last chunk, the expected number of adjacency-lists kept is at least $\mu/8$.*

[1]The last chunk of the Euler tour only visits $((2 \cdot n' - 1) \bmod \mu) + 1$ vertices where n' denotes the number of vertices in the connected component of the starting node s.

Proof. Let $R = (v_1, \ldots, v_\mu)$ be the sequence of vertices visited by an arbitrary chunk $\mathcal{R}$ of the Euler tour $\mathcal{T}$, excluding the last chunk. Let a be the number of entries in R that represent first or last visits of inner-tree vertices from the spanning tree T_s on $\mathcal{T}$. These a entries account for an expected number of $a/2$ adjacency-lists actually stored and kept in $\mathcal{R}$. Note that if for some vertex $v \in \mathcal{T}$ both its first and last visit happen within $\mathcal{R}$, then v's adjacency-list is kept with probability one. Similarly, if there are any visits of leaf nodes from T_s within $\mathcal{R}$, then their adjacency-lists are kept for sure; let b denote the number of these leaf node entries in R. What remains are $\mu - a - b$ *intermediate* (neither first nor last) visits of vertices within $\mathcal{R}$; they do not contribute any additional adjacency-lists.

We can bound $\mu - a - b$ using the observation that any intermediate visit of a tree node v on $\mathcal{T}$ is preceded by a last visit of a child v' of v and proceeded by a first visit of another child v'' of v. Thus, $\mu - a - b \leq \lceil \mu/2 \rceil$, that is $a + b \geq \lfloor \mu/2 \rfloor$, which implies that the expected number of distinct adjacency-lists being kept for $\mathcal{R}$ is at least $\lfloor \mu/2 \rfloor / 2 \geq \mu/8$. ∎

4. The Dynamic Incremental Algorithm

In this section we concentrate on the incremental version for sparse graphs with $\Theta(n)$ updates where each update inserts an edge. Thus, BFS levels can only decrease over time. Before we start, let us fix some notation: for $i \geq 1$, $G_i = (V, E_i)$ is to denote the graph after the i-th update, G_0 is the initial graph. Let $d_i(v)$, $i \geq 0$, stand for the BFS level of node v if it can be reached from the source node s in G_i and n otherwise. Furthermore, for $i \geq 1$, let $\Delta d_i(v) = |d_{i-1}(v) - d_i(v)|$. The main ideas of our approach are as follows:

Checking Connectivity; Type A updates. In order to compute the BFS levels for G_i, $i \geq 1$, we first run an EM connected components algorithm (for example the one in [13] taking $O(\mathrm{sort}(n) \cdot \log B)$ I/Os) in order to check, whether the insertion of the i-th edge (u, v) enlarges the connected component $\mathcal{C}_s$ of the source vertex s. If yes (let us call this a *Type A update*), then w.l.o.g. let $u \in \mathcal{C}_s$ and let $\mathcal{C}_v$ be the connected component that comprises v. The new edge (u, v) is then the only connection between the existing BFS-tree for s and $\mathcal{C}_v$. Therefore, we can simply run MR_BFS on the subgraph G' defined by the vertices in $\mathcal{C}_v$ with source v and add $d_{i-1}(u) + 1$ to all distances obtained. This takes $O(n_v + \mathrm{sort}(n))$ I/Os where n_v denotes the number of vertices in $\mathcal{C}_v$.

If the i-th update does not merge $\mathcal{C}_s$ with some other connected component but adds an edge within $\mathcal{C}_s$ (*Type B update*) then we need to do something more fancy:

Dealing with small changes; Type B updates. Now for computing the BFS levels for G_i, $i \geq 1$, we pre-feed the adjacency-lists into a sorted pool $\mathcal{H}$ according to the BFS levels of their respective vertices in G_{i-1} using a certain advance $\alpha > 1$, i.e., the adjacency list for v is added to $\mathcal{H}$ when creating BFS level $\max\{0, d_{i-1}(v) - \alpha\}$ of G_i. This can be done I/O-efficiently as follows. First we extract the adjacency-lists for vertices having BFS levels up to α in G_{i-1} and put them to $\mathcal{H}$ where they are kept sorted by node indices. From the remaining adjacency-lists we build a sequence $\mathcal{S}$ by sorting them according to BFS levels in G_{i-1} (primary criterion) and node indices (secondary criterion). For the construction of each new BFS level of G_i we merge a subsequence of $\mathcal{S}$ accounting for one BFS level in G_{i-1} with $\mathcal{H}$ using simple scanning.

Therefore, if $\Delta d_i(v) \leq \alpha$ for all $v \in V$ then all adjacency-lists will be added to $\mathcal{H}$ in time and can be consumed from there without random I/O. Each adjacency-list is scanned

at most once in $\mathcal{S}$ and at most α times in $\mathcal{H}$. Thus, if $\alpha = o(\sqrt{B})$ this approach causes less I/O than MM_BFS.

Dealing with larger changes. Unfortunately, in general, there may be vertices v with $\Delta d_i(v) > \alpha$. Their adjacency-lists are not prefetched into $\mathcal{H}$ early enough and therefore have to be imported into $\mathcal{H}$ using random I/Os to whole clusters just like it is done in MM_BFS. However, we apply the modified clustering procedure described in Section 3.2 on G_{i-1}, the graph without the i-th new edge (whose connectivity is the same as that of G_i) with chunk size $\alpha/4$.

Note that this may result in $\Theta(n/\alpha)$ cluster accesses, which would be prohibitive for small α. Therefore we restrict the number of random cluster accesses to $\alpha \cdot n/B$. If the dynamic algorithm does not succeed within these bounds then it increases α by a factor of two, computes a new clustering for G_{i-1} with larger chunk size and starts a *new attempt* by repeating the whole approach with the increased parameters. Note that we do not need to recompute the spanning tree for the for the second, third, ... attempt.

At most $O(\log B)$ attempts per update. The j-th attempt, $j \geq 1$, of the dynamic approach to produce the new BFS-level decomposition will apply an advance of $\alpha_j := 32 \cdot 2^j$ and recompute the modified clustering for G_{i-1} using chunk size $\mu_j := 8 \cdot 2^j$. Note that there can be at most $O(\log \sqrt{B}) = O(\log B)$ failing attempts for each edge update since by then our approach allows sufficiently many random accesses to clusters so that all of them can be loaded explicitly resulting in an I/O-bound comparable to that of static MM_BFS. In Section 5, however, we will argue that for most edge updates within a longer sequence, the advance value and the chunk size value for the succeeding attempt are bounded by $O(B^{1/3})$ implying significantly improved I/O performance.

Restricting waiting time in $\mathcal{H}$. There is one more important detail to take care of: when adjacency-lists are brought into $\mathcal{H}$ via explicit cluster accesses (because of insufficient advance α_j in the prefetching), these adjacency-lists will re-enter $\mathcal{H}$ once more later on during the (for these adjacency-lists by then useless) prefetching. Thus, in order to make sure that unnecessary adjacency-lists do not stay in $\mathcal{H}$ forever, each entry in $\mathcal{H}$ carries a time-stamp ensuring that superfluous adjacency-lists are evicted from $\mathcal{H}$ after at most $\alpha_j = O(2^j)$ BFS levels.

Lemma 4.1. *For sparse graphs with $O(n)$ updates, each Type B update succeeding during the j-th attempt requires $O(2^j \cdot n/B + \mathrm{sort}(n) \cdot \log B)$ I/Os.*

Proof. Deciding whether a Type B update takes place essentially requires a connected components computation, which accounts for $O(\mathrm{sort}(n) \cdot \log B)$ I/Os. Within this I/O bound we can also compute a spanning tree T_s of the component holding the starting vertex s but excluding the new edge. Subsequently, there are $j = O(\log B)$ attempts, each of which uses $O(\mathrm{sort}(n))$ I/Os to derive a new modified clustering based on an Euler tour with increasing chunk sizes around T_s. Furthermore, before each attempt we need to initialize $\mathcal{H}$ and $\mathcal{S}$, which takes $O(\mathrm{sort}(n))$ I/Os per attempt. The worst-case number of I/Os to (re-) scan adjacency-lists in $\mathcal{H}$ or to explicitly fetch clusters of adjacency-lists doubles after each attempt. Therefore it asymptotically suffices to consider the (successful) last attempt j, which causes $O(2^j \cdot n/B)$ I/Os. Furthermore, each attempt requires another $O(\mathrm{sort}(n))$ I/Os to pre-sort explicitly loaded clusters before they can be merged with $\mathcal{H}$ using a single scan just like in MM_BFS. Adding all contributions yields the claimed I/O bound of $O(2^j \cdot n/B + \mathrm{sort}(n) \cdot \log B)$ for sparse graphs. ∎

5. Analysis

We split our analysis of the incremental BFS algorithm into two parts. The first (and easy one) takes care of Type A updates:

Lemma 5.1. *For sparse undirected graphs with $\Theta(n)$ updates, there are at most $n-1$ Type A updates causing $O(n \cdot \mathrm{sort}(n) \cdot \log B)$ I/Os in total.*

Proof. Each Type A update starts with an EM connected components computation causing $O(\mathrm{sort}(n) \cdot \log B)$ I/Os per update. Since each node can be added to the connected component $\mathcal{C}_s$ holding the starting vertex s only once, the total number of I/Os spend in calls to the MR-BFS algorithm on components to be merged with $\mathcal{C}_s$ is $O(n + \mathrm{sort}(n))$. Producing the output takes another $O(\mathrm{sort}(n))$ per update. ∎

Now we turn to Type B updates:

Lemma 5.2. *For sparse undirected graphs with $\Theta(n)$ updates, all Type B updates cause $O(n \cdot (n^{2/3} + \mathrm{sort}(n) \cdot \log B))$ I/Os in total with high probability.*

Proof. Recall that $d_i(v)$, $i \geq 0$, stands for the BFS level of node v if it can be reached from the source node s in G_i and n otherwise. If upon the i-th update the dynamic algorithm issues an explicit fetch for the adjacency-lists of some vertex v kept in some cluster C then this is because $\Delta d_i(v) = d_{i-1}(v) - d_i(v) > \alpha$ for the current advance α. Note that for all other vertices $v' \in C$, there is a path of length at most μ in G_{i-1}, implying that $|d_{i-1}(v') - d_{i-1}(v)| \leq \mu$ as well as $|d_i(v) - d_i(v')| \leq \mu$. Having current chunk size $\mu = \alpha/4$, this implies

$$\begin{aligned} \Delta d_i(v') &= d_{i-1}(v') - d_i(v') \\ &= d_{i-1}(v') - d_{i-1}(v) + d_{i-1}(v) - d_i(v) + d_i(v) - d_i(v') \\ &> \alpha - 2\mu \\ &\geq \alpha/2. \end{aligned}$$

If the i-th update needs j attempts to succeed then, during the (failing) attempt $j-1$, it has tried to explicitly access $\alpha_{j-1} \cdot n/B + 1$ distinct clusters. Out of these at least $\alpha_{j-1} \cdot n/B = 2^{j+4} \cdot n/B$ clusters carry an expected amount of at least $\mu_{j-1}/8 = 2^{j-1}$ adjacency-lists each. This accounts for an expected number of at least $2^{2 \cdot j+3} \cdot n/B$ distinct vertices, each of them featuring $\Delta d_i(\cdot) \geq \alpha_{j-1}/2 = 2^{j+3}$. With probability at least $1/2$ we actually get at least half of the expected amount of distinct vertices/adjacency-lists, i.e., $2^{2 \cdot j+2} \cdot n/B$. Therefore, using the definitions $D_i = \sum_{v \in V \setminus \{s\}} d_i(v)$ and $\Delta D_i = |D_{i-1} - D_i|$, if the i-th update succeeds within the j-th attempt we have $\Delta D_i \geq 2^{3 \cdot j+5} \cdot n/B =: Y_j$ with probability at least $1/2$. Let us call this event *a large j-yield*.

Since each attempt uses a new clustering with independent choices for $r(\cdot)$, if we consider two updates i' and i'' that succeed after the same number of attempts j, then both i' and i'' have a large yield with probability at least $1/2$, independent of each other. Therefore, we can use Chernoff bounds [10] in order to show that out of $k \geq 16 \cdot c \cdot \ln n$ updates that all succeed within their j-th attempt, at least $k/4$ of them have a large j-yield with probability at least $1 - n^{-c}$ for an arbitrary positive constant c. Subsequently we will prove an upper bound on the total number of large j-yields that can occur during the whole update sequence.

The quantity ΔD_i provides a global measure as for how much the BFS levels change after inclusion of the i-th edge from the update sequence. If there are $m' = \Theta(n)$ edge inserts in total, then

$$n^2 > D_0 \geq D_1 \geq \ldots \geq D_{m'-1} \geq D_{m'} > 0.$$

A large j-yield means $\Delta D_i \geq Y_j$. Therefore, in the worst case there are at most $n^2/Y_j = n^2/(2^{3\cdot j+5} \cdot n/B) = n \cdot B/2^{3\cdot j+5}$ large j-yield updates and – according to our discussion above – it needs at most $k_j := 4 \cdot n \cdot B/2^{3\cdot j+5}$ updates that succeed within the j-th attempt to have at least $k_j/4$ large j-yield updates with high probability[2].

For the last step of our analysis we will distinguish two kinds of Type B updates: those that finish using an advance value $\alpha_{j^*} < B^{1/3}$ (Type B1), and the others (Type B2). Independent of the subtype, an update costs $O(\alpha_{j^*} \cdot n/B + \mathrm{sort}(n) \cdot \log B) = O(2^{j^*} \cdot n/B + \mathrm{sort}(n) \cdot \log B)$ I/Os by Lemma 4.1. Obviously, for an update sequence of $m' = \Theta(n)$ edge insertions there can be at most $\Theta(n)$ updates of Type B1, each of them accounting for at most $O(n/B^{2/3} + \mathrm{sort}(n) \cdot \log B)$ I/Os. As for Type B2 updates we have already shown that with high probability there are at most $O(n \cdot B/2^{3\cdot j^*})$ updates that succeed with advance value $\Theta(2^{j^*})$. Therefore, using Boole's inequality, the total amount of I/Os for all Type B2 updates is bounded by

$$O\left(\left(\sum_{g\geq 0} \frac{n \cdot B}{(B^{1/3} \cdot 2^g)^3} \cdot \frac{B^{1/3} \cdot 2^g \cdot n}{B}\right) + n \cdot \mathrm{sort}(n) \cdot \log B\right) =$$

$O(n \cdot (n/B^{2/3} + \mathrm{sort}(n) \cdot \log B))$ with high probability. ∎

Combining the two lemmas of this section implies

Theorem 5.3. *For general sparse undirected graphs of initially n nodes and $O(n)$ edges and $\Theta(n)$ edge insertions, dynamic BFS can be solved using amortized $O(n/B^{2/3} + \mathrm{sort}(n) \cdot \log B)$ I/Os per update with high probability.*

6. Decremental Version and Extensions.

Having gone through the ideas of the incremental version, it is now close to trivial to come up with a symmetric external-memory dynamic BFS algorithm for a sequence of edge deletions: instead of pre-feeding adjacency-lists into using an *advance* of α_j levels, we now apply a *lag* of α_j levels. Therefore, the adjacency-list for a vertex v is found in $\mathcal{H}$ as long as the deletion of the i-th edge does not increase $d_i(v)$ by more than α. Otherwise, an explicit random access to the cluster containing v's adjacency-list is issued later on. All previously used amortization arguments and bounds carry through, the only difference being that $d_i(\cdot)$ values may monotonically increase instead of decrease.

Better amortized bounds can be obtained if $\omega(n)$ updates take place and/or G_0 has $\omega(n)$ edges. Then we have the potential to amortize more random accesses per attempt, which leads to larger j-yields and reduces the worst-case number of expensive updates. Consequently, we can reduce the defining threshold between Type B1 and Type B2 updates,

[2]We also need to verify that $k_j \geq 16 \cdot c \cdot \ln n$. As observed before, the dynamic algorithm will not increase its advance and chunk size values beyond $O(\sqrt{B})$ implying $2^j = O(\sqrt{B})$. But then we have $k_j = 4 \cdot n \cdot B/2^{3\cdot j+5} = \Omega(n/\sqrt{B})$ and $n/\sqrt{B} \geq n/\sqrt{M} \geq n/\sqrt{n} \geq 16 \cdot c \cdot \ln n$ for sufficiently large n.

thus eventually yielding better amortized I/O bounds. Details we be provided in the full version of this paper.

Modifications along similar lines are in order if external-memory is realized by flash disks [9]: compared to hard disks, flash memory can sustain many more unstructured read I/Os per second but on the other hand flash memory usually offers less read/write bandwidth than hard disks. Hence, in algorithms like ours that are based on a trade-off between unstructured read I/Os and bulk read/write I/Os, performance can be improved by allowing more unstructured read I/Os (fetching clusters) if this leads to less overall I/O volume (scanning hot pool entries).

7. Conclusions

We have given the first non-trivial external-memory algorithm for dynamic BFS. Even though we obtain significantly better I/O bounds than for the currently best static algorithm, there are a number of open problems: first of all, our bounds dramatically deteriorate for mixed update sequences (edge insertions and edge deletions in arbitrary order and proportions); besides oscillation effects, a single edge deletion (insertion) may spoil a whole chain of amortizations for previous insertions (deletions). Also, it would be interesting to see, whether our bounds can be further improved or also hold for shorter update sequences. Finally, it would be nice to come up with a deterministic version of the modified clustering.

Acknowledgements

We would like to thank Deepak Ajwani for very helpful discussions.

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM, 31(9)*, pages 1116–1127, 1988.

[2] L. Arge, G. Brodal, and L. Toma. On external-memory MST, SSSP and multi-way planar graph separation. In *Proc. 8th Scand. Workshop on Algorithmic Theory (SWAT)*, volume 1851 of *LNCS*, pages 433–447. Springer, 2000.

[3] M. Atallah and U. Vishkin. Finding Euler tours in parallel. *Journal of Computer and System Sciences, 29(30)*, pages 330–337, 1984.

[4] A. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. Westbrook. On external memory graph traversal. In *Proc. 11th Ann. Symposium on Discrete Algorithms (SODA)*, pages 859–860. ACM-SIAM, 2000.

[5] Y. J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamasia, D. E. Vengroff, and J. S. Vitter. External memory graph algorithms. In *Proc. 6th Ann.Symposium on Discrete Algorithms (SODA)*, pages 139–149. ACM-SIAM, 1995.

[6] T. H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1990.

[7] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. *In 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 714–723, 2006.

[8] D. Eppstein, Z. Galil, and G. Italiano. Dynamic graph algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 8. CRC Press, 1999.

[9] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37:138–163, 2005.

[10] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990.

[11] K. Mehlhorn and U. Meyer. External-memory breadth-first search with sublinear I/O. In *Proc. 10th Ann. European Symposium on Algorithms (ESA)*, volume 2461 of *LNCS*, pages 723–735. Springer, 2002.

[12] U. Meyer, P. Sanders, and J. Sibeyn (Eds.). *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS*. Springer, 2003.
[13] K. Munagala and A. Ranade. I/O-complexity of graph algorithms. In *In Proc. 10th Ann. Symposium on Discrete Algorithms (SODA)*, pages 687–694. ACM-SIAM, 1999.
[14] L. Roditty. *Dynamic and static algorithms for path problems in graphs*. PhD thesis, Tel Aviv University, 2006.
[15] J. D. Ullman and M. Yannakakis. High-probability parallel transitive closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, February 1991.
[16] J. S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM computing Surveys, 33*, pages 209–271, 2001. Revised version (August 2007) available online at `http://www.cs.purdue.edu/homes/jsv/Papers/Vit.IO_survey.pdf`.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 561-572
www.stacs-conf.org

ANALYTIC ASPECTS OF THE SHUFFLE PRODUCT

MARNI MISHNA [1] AND MIKE ZABROCKI [2]

[1] Department of Mathematics, Simon Fraser University, Burnaby, Canada
E-mail address: mmishna@sfu.ca

[2] Department of Mathematics and Statistics, York University, Toronto, Canada
E-mail address: zabrocki@mathstat.yorku.ca

Abstract. There exist very lucid explanations of the combinatorial origins of rational and algebraic functions, in particular with respect to regular and context free languages. In the search to understand how to extend these natural correspondences, we find that the shuffle product models many key aspects of D-finite generating functions, a class which contains algebraic. We consider several different takes on the shuffle product, shuffle closure, and shuffle grammars, and give explicit generating function consequences. In the process, we define a grammar class that models D-finite generating functions.

Introduction

Generating functions of languages

The (ordinary) generating function of a language $\mathcal{L}$ is the sum

$$L(z) = \sum_{w \in \mathcal{L}} z^{|w|},$$

where $|w|$ is the length of the word. This sum is a formal power series if there are finitely many words of a given length. In this case, we say the language is *proper*, and we can rewrite $L(z)$ as $L(z) = \sum \ell(n) z^n$, where $\ell(n)$ is the number of words in L of length n. In the case where we have an unambiguous grammar to describe a regular language or a context free language, one can automatically generate equations satisfied by generating function directly from the grammar. These are the well known translations:

$$\begin{array}{lll} \mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2 & \Longrightarrow & L(z) = L_1(z) + L_2(z) \\ \mathcal{L} = \mathcal{L}_1 \cdot \mathcal{L}_2 & \Longrightarrow & L(z) = L_1(z) L_2(z) \\ \mathcal{L} = \mathcal{L}_1^* & \Longrightarrow & L(z) = (1 - L_1(z))^{-1}. \end{array}$$

Generating functions of formal languages are now a very established tool for algorithm analysis (see [12] for many references) and increasingly for random generation [9]. In this context, we are also interested in the *exponential generating function* of a language. The

1998 ACM Subject Classification: F.4.3 Formal Languages.
Key words and phrases: generating functions, formal languages, shuffle product.

two are related by the Laplace-Borel transform, however it is sufficient for our purposes to think of the exponential generating function $\hat{L}(z)$ as the Hadamard product of $L(z)$ and $\exp(z) = \sum \frac{z^n}{n!}$; that is, $\hat{L}(z) = \sum \ell(n)\frac{z^n}{n!}$.

One spectacular feature of generating functions of languages is the extent to which their analytic complexity models the complexity of the language. Specifically, we have the two classic results: first, regular languages have rational generating functions, and second, those context-free languages which are not inherently ambiguous have an algebraic generating function. The context-free languages form a large and historically important subclass of all objects which have algebraic generating functions. Bousquet-Mélou provides us [6, 7] with an interesting discussion of the nature of combinatorial structures that possess algebraic and rational generating functions, including broad classes that are not representable as context-free languages.

There remain unanswered questions related to other classes of languages, and other classes of functions. An example of the former is the question of Flajolet [10]: "In which class of transcendental functions do generating functions of (general) context free languages lie?" An example of the latter is the identification of languages whose generating functions are *D-finite*[1]. This is an exceptional class of functions [24], which, for the moment, lacks a satisfying combinatorial explanation. We survey some current understandings in Section 1.3, and provide a language theoretic interpretation of one in Section 3.1.

To capture the analytic complexity of D-finite generating functions we should not expect a simple climbing of the language hierarchy (to indexed or context sensitive, say), as there are different notions of complexity in competition. For example the language $\{a^n b^n c^n : n \in \mathbb{N}\}$ is difficult to recognize, but trivial to enumerate. Likewise, the generating function of the relatively simple looking language $\{z^{n^2} : n \in \mathbb{N}\}$ has a natural boundary at $|z| = 1$, which is a trademark of very complex analytic behaviour.

The shuffle product

In the absence of the obvious answers, we consider a very common, and useful operator, the *shuffle product*, and discover that it fills in many interesting holes in this story. Consider the words w, uw_1 and vw_2, and the letters $u, v \in \Sigma$. We define the shuffle product of two words recursively by the equation

$$uw_1 \sqcup\!\sqcup vw_2 = u(w_1 \sqcup\!\sqcup vw_2) + v(uw_1 \sqcup\!\sqcup w_2), \quad w \sqcup\!\sqcup \epsilon = w; \quad \epsilon \sqcup\!\sqcup w = w.$$

Here the union is disjoint, and we distinguish duplicated letters from the second word by a bar: $a \sqcup\!\sqcup a = \{\overline{a}a, a\overline{a}\}$. Using the shuffle product we can define a class of languages with associated generating functions that form a class that strictly contains algebraic functions; it allows us to model a very straightforward combinatorial interpretation of the derivative (indeed in some interesting non-commutative algebras the shuffle product is even called a derivative); and it allows us to neatly consider some larger classes which are simultaneously more complex from the language and generating function points of view.

[1]D-finite, also known as holonomic, functions satisfy linear differential equations with polynomial coefficients.

Goal and Results

The aim of this study is two-fold. We hope that a greater understanding of generating function implications of adding the shuffle product to context free languages provides insight to a larger class of combinatorial problems. The second goal is to understand the combinatorial interpretations of different function classes that arise between algebraic and D-finite. The shuffle is a natural combinatorial product to consider since it is, in some sense, a generalization of pointing.

In the present work, we first examine the shuffle as an operator *on* languages, and in the second part we consider the shuffle as a grammar production rule *to define* languages. We show that the shuffle closure of the context free languages is D-finite; we give the asymptotic growth of coefficients of two classes using shuffle; we define a special pointing class that describes all D-finite functions; and discuss the shuffle closure of a language.

In the next section we review interpretations of differential equations. This is followed by a discussion on the shuffle of languages, and some descriptions of shuffle grammars.

1. Interpreting differential equations combinatorially

1.1. The class of D-finite functions

The class of D-finite functions is of interest to the combinatorialist for many reasons. The coefficient sequence of a D-finite power series is P-recursive: it satisfies a linear recurrence of fixed length with polynomial coefficients, and hence is easy to generate, manipulate, and even "guess" their form. By definition, D-finite functions satisfy linear differential equations with polynomial coefficients, and thus it is relatively straightforward in many cases to perform an asymptotic analysis on the coefficients, even without a closed form for the generating function. One important feature that we use here is that a P-recursive sequence grows asymptotically like

$$\ell(n) \sim \lambda (n!)^{r/s} \exp(Q(n^{1/m}\omega^n n^\alpha (\log n)^k))$$

where $r, s, m, n, k \in \mathbb{N}$, Q is a polynomial and λ, ω, α, are complex numbers. We contrast this to the asymptotic template satisfied by coefficients of algebraic functions:

$$\ell(n) \sim \kappa \frac{n^d}{\Gamma(d+1)} \omega^{-n}, \tag{1.1}$$

where κ is an algebraic number and $d \in \mathbb{Q} \setminus \{-1, -2, \dots\}$. (A very complete source on the theory of asymptotic expansions of coefficients of algebraic functions arising in the combinatorial context is [12, Section VII.4.1].) Notable differences include the exponential/logarithmic factors, the power of a factorial, and the allowable exponents of n.

We shall use the following properties of the D-finite functions: The function $1/f$ is D-finite if, and only if, f is of the form $\exp(g)h$, where g and h are algebraic [23]; The Hadamard product $f \times g = \sum f_n g_n z^n$ of two D-finite functions $f = \sum f_n z^n$ and $g = \sum g_n z^n$ is also D-finite.

1.2. The simplest shuffle: the point

Pointing (or marking) is an operation that has been long studied in connection with structures generated by grammars. The point of an word w, denoted $P(w)$, is a set of words, each with a different position marked. For example, $P(abc) = \{\overline{a}bc, a\overline{b}c, ab\overline{c}\}$. From the enumerative point of view we remark that the two languages L, and $\mathcal{L}_1 = P(\mathcal{L}) = \{P(w) : w \in \mathcal{L}\}$, satisfies the enumerative relation

$$\ell_1(n) = n\ell(n), \tag{1.2}$$

and hence $L_1(z) = z\frac{d}{dz}L(z)$. The pointing operator is relevant to our discussion because of the simple bijective correspondence between $P(\mathcal{L})$ and $\mathcal{L} \sqcup\!\sqcup a = \{w \sqcup\!\sqcup a : w \in \mathcal{L}\}$.

The first obvious question is, "does pointing increase expressive power?". In the case of regular languages and context free languages the answer is no; We can add a companion non-terminal for each non terminal that generates a language isomorphic to the pointed language. Let $\overline{A}$ be the pointed version of A. We add the following rules which model pointing:

$$\overline{(AB)} = \overline{A}B + A\overline{B}, \quad \overline{(A+B)} = \overline{A} + \overline{B}$$

Remark how these rules resemble the corresponding product and sum rules for differentiation. Furthermore, from the point of view of generating functions, we know that the derivative of a rational function is rational again, and the derivative of an algebraic function is again algebraic, and so we know immediately that we could not hope to increase the class of generating functions represented by this method.

Pointing, when paired with a "de-pointing" operator which removes such marks, becomes powerful enough to describe other kinds of constructions, namely labelled cycles and sets [13, 15]. In this case we can describe set partitions, and which has exponential generating function $\exp(\exp(z) - 1)$, which is not D-finite.

It takes much more effort [5] to define a pointing operator with a differentiation property as in Eq.(1.2) for unlabelled structures defined using Set and Cycle constructions. It is a fruitful exercise, as one can then generate approximate size samplers with expected linear time complexity.

1.3. Other combinatorial derivatives

Combinatorial species theory [2] provides a rich formalism for explaining the interplay between analytic and combinatorial representions of objects. In particular, using the vehicle of the the cycle index series, and there are several possibilities on how to relate them to (multivariate) D-finite functions [18, 21]. In this realm, given any arbitrary linear differential equation with polynomial coefficients we can define a set of grammar operators that allow us to construct a pair of species whose difference has a generating function that satisfies the given differential equation. Unfortunately at present we lack the intuition to understand what this class "is", specifically, we lack the tools to construct a test to see if any given class or language falls within it.

In Section 3.4 we give a language theoretic interpretation of the derivative of a species; specifically a grammar system, from which, for any linear differential equation with coefficients from $\mathbb{Q}[x]$ we can generate a language whose generating function satisfies this equation.

1.4. Other differential classes

There are several other natural function classes related to the differential equations. A series $f(z) \in K[[t]]$ is said to be *constructible differentiably algebraic (CDF)* if it belongs to some finitely generated ring which is closed under differentiation. [3, 4]. This is equivalent to satisfying a system of differential equations of a given form. Combinatorially, any CDF function can be interpreted as a family of enriched trees. Theorem 3 of [3] gives the result that if $\sum a_n/n!t^n$ is CDF, then $|a_n| = O(\alpha^n n!)$ for some complex constant α. This class is not closed under Hadamard product, and any arbitrary CDF function is unlikely to have the image under the Borel transform also CDF. This is the key closure property required for a meaningful correspondence with respect to the shuffle product.

A larger class which contains both CDF and D-finite is differentiably algebraic. A function is *differentiably algebraic (DA)* if it satisfies an algebraic differential equation of the form $P(x, y, y', \ldots, y^{(n)}) = 0$ where P is a non-trivial polynomial in its $n + 2$ variables. (See Rubel's survey [22] for many references.)

The set of DA functions is closed under multiplicative inverse and Hadamard product. These two facts together are sufficient to prove that *all of the classes we consider are differentiably algebraic.*

1.5. Generating functions and shuffles

Generating functions are useful tool for the automatic studies of certain combinatorial problems. The shuffle operator has a straightforward implication on the generating function, as we shall see.

With the aid of the shuffle product, Flajolet *et al.* [11] are able to perform a straightforward analysis of four problems in random allocation. By using some systematic translations, they are able to derive integral representations for expectations and probability distributions. As they remark, the shuffle of languages appears in several places relating to analysis of algorithms (such as evolution of two stacks in a common memory area).

2. The shuffle of two languages

The shuffle of two languages is defined as

$$\mathcal{L}_1 \sqcup\!\sqcup \mathcal{L}_2 = \bigcup_{w_1 \in \mathcal{L}_1, w_2 \in \mathcal{L}_2} (w_1 \sqcup\!\sqcup w_2).$$

In order to use a generating function approach, we assume that $\mathcal{L}_1$ is a language over the alphabet Σ_1, and $\mathcal{L}_2$ is a language over Σ_2, and $\Sigma_1 \cap \Sigma_2 = \emptyset$. If they share an alphabet, it suffices to add a bar on top of the copy from Σ_2.

2.1. The shuffle closure of context free languages

We consider the shuffle closure of a language in the next section, and first concentrate on the shuffle closure of a class of languages. For any given class of languages $\mathcal{C}$, the shuffle closure can be defined recursively as the (infinite) union of $S_0, S_1, \ldots$, the sequence recursively defined by

$$S_0 = \mathcal{C}, \qquad S_n = \{\mathcal{L}_1 \sqcup\!\sqcup \mathcal{L}_2 : \mathcal{L}_1 \in S_{n-1}, \mathcal{L}_2 \in \mathcal{C}\}.$$

The shuffle product is commutative and associative [20], and thus the closure contains $S_j \sqcup\!\sqcup S_i$, for any i and j. Remark, that for any given language in the closure, there is a bound on the number of shuffle productions that can occur in any derivation tree; namely, if $\mathcal{L} \in S_n$, that bound is n.

In general, we denote the closure of a class of languages under shuffle as $\mathcal{C}^{\sqcup\!\sqcup}$. The class of regular languages is closed under the shuffle product, since the shuffle of any two regular languages is regular. However, the context free languages are not closed under the shuffle product [20], and hence we consider its closure.

The prototypical language in this class is the shuffle of (any finite number of) Dyck languages. Let $|w|_a$ be count the number of occurences of the letter a in the word w. Let $\mathcal{D}$ be the Dyck language over the alphabet $\Sigma = \{u, d\}$:

$$\mathcal{D} = \{w \in \Sigma^* : w'v = w \implies |w'|_u \geq |w'|_d \text{ and } |w|_u = |w|_d.\}$$

We construct an isomorphic version $\mathcal{E}$, over the alphabet $\{l, r\}$.

The language $\mathcal{D} \sqcup\!\sqcup \mathcal{E}$ has encodes random walks restricted to the quarter plane with steps from u(p), d(own), r(ight), and l(eft) that return to the origin. By considering the larger language of Dyck prefixes, we can models walks that end anywhere in the quarter plane. Indeed, as the shuffle does preserve two distinct sets of prefix conditions, there are many examples of random walks in bounded regions that can be expressed as shuffles of algebraic languages.

It might be interesting to consider other standard questions of classes of languages for this closure class; in particular if interesting random walks arise.

2.2. The closure is D-finite

In order to show that the shuffle product of two languages with D-finite generating functions also has a D-finite generating function, we consider the following classic observation on the enumeration of shuffles of languages.

If $\mathcal{L}$ is the shuffle of $\mathcal{L}_1$ and $\mathcal{L}_2$, then the number of words of length n in $\mathcal{L}$ are easily counted if the generating series for $L_1(z) = \ell_1(n)z^n$ and $L_2(z) = \ell_2(n)z^n$ are known by the following formula:

$$\ell(n) = \sum_{n_1+n_2=n} \binom{n}{n_1\, n_2} \ell_1(n_1)\ell_2(n_2).$$

To see this, recognize that a word in $\mathcal{L}$ is a composed of two words, and a set of positions for the letters in the word from $\mathcal{L}_1$, This is equivalent to

$$\frac{\ell(n)}{n!} = \sum_{n_1+n_2=n} \frac{\ell_1(n_1)}{n_1!} \frac{\ell_2(n_2)}{n_2!}, \tag{2.1}$$

which amounts to the relation between the *exponential generating functions* of the three languages:

$$\mathcal{L} = \mathcal{L}_1 \sqcup\!\sqcup \mathcal{L}_2 \qquad \implies \qquad \hat{L}(z) = \hat{L}_1(z)\hat{L}_2(z). \tag{2.2}$$

Using these relations, we can easily prove the following result.

Proposition 2.1. *If $\mathcal{L}_1$ and $\mathcal{L}_2$ are languages with D-finite ordinary generating functions, then the generating series for $\mathcal{L} = \mathcal{L}_1 \sqcup\!\sqcup \mathcal{L}_2$, $L(z)$ is also D-finite.*

As is the case with many of the most interesting closure properties of D-finite functions, the proof follows from the closure of D-finite functions under Hadamard product [19].

Proof. Since D-finite functions are closed under Hadamard product, the ordinary generating function is D-finite if and only if the exponential generating function of a sequence is D-finite. Consequently, if $L_1(z)$ and $L_2(z)$ are D-finite, then so are the exponential generating functions, $\hat{L_1}(z)$ and $\hat{L_2}(z)$. By closure under product, $\hat{L}(z)$ is D-finite, and thus so is $L(z)$. ∎

This result has the following consequences.

Corollary 2.2. *If $\mathcal{L}_1$ and $\mathcal{L}_2$ are context free languages which are not inherently ambiguous, then the generating series $L(z)$ for $\mathcal{L} = \mathcal{L}_1 \sqcup\!\sqcup \mathcal{L}_2$ is D-finite.*

Corollary 2.3. *Any language in the shuffle closure of context free languages has a D-finite generating function.*

2.3. Asymptotic template for $\ell(n)$

We continue the example from the previous section using the two Dyck languages $\mathcal{D}$ and $\mathcal{E}$. It is straightforward to compute that $D(z) = E(z) = \sum \binom{2n}{n} \frac{1}{n+1} z^n$. Thus, $\ell(n)$, the number of words of length n in the shuffle is given by

$$\ell(n) = \sum_{n_1+n_2=n} \binom{n}{n_1} \binom{n_1}{n_1/2} \binom{n_2}{n_2/2}.$$

We remark that an asymptotic expression for $\ell(n)$ can be determined by first using the Vandermonde-Chu identity to simplify $\ell(n)$:

$$\ell(n) = \binom{n}{\lfloor n/2 \rfloor} \binom{n+1}{\lceil n/2 \rceil},$$

and then by applying Stirling's formula. Since $\ell(n) \sim 4^n/n$, we see that it the resulting series is not algebraic. Flajolet uses this technique extensively in [10] to prove that certain context-free languages are inherently ambiguous. Thus, we have that our class has generating functions strictly contains the algebraic functions.

Thus, we have some elements of a class of function with a nice asymptotic expansion. A rough calculation gives that that the shuffle of two languages, with respective asymptotic growth of $\kappa_i n^{r_i} (\alpha_i)^n$, for $i = 1, 2$ respectively, is given by the expression

$$\ell(n) \sim \kappa n^{r_1+r_2} (\alpha_1 + \alpha_2 - r_1 - r_2)^n.$$

How could one hope to prove directly that all elements in this class have an expansion of the form

$$\ell(n) \sim \kappa \alpha^n n^r,$$

where now r can be *any* rational, and κ is no longer restricted to algebraic numbers? It seems that it should be possible to prove this at least for the shuffles of series which satisfy the hypotheses of Theorem 3.11 [7], using a more generalized form of the Chu-Vandermonde identity, or for the closure of the sub-class of context-free languages posessing an $\mathbb{N}$-algebraic generating function. In this case the $d = -3/2$, and this simplifies the analyses considerably. Unfortunately, it does not seem like a direct application of Bender's method [12, Theorem VI.2] applies.

Theorem 3.2 states that the asymptotic form will not contain any powers of $n!$ greater than 2. This illustrates a limitation with the expressive power of the shuffle closure of context free languages: there are known natural combinatorial objects which have D-finite generating functions with coefficients that grow asymptotically with higher powers of $n!$.

For example, the number of k-regular graphs for $k > 4$ contains $(n!)^{5/2}$, and the conjectured asymptotic for for k-uniform Young tableaux [8] contains $n!^{k/2-1}$.

3. Shuffle grammars

We extend the first approach by allowing the shuffle to come into play earlier in the story; we add the shuffle operator to our grammar rewriting rules. Shuffle grammars as defined by Gischer [14] include a shuffle rule, and a shuffle closure rule. We consider these in Section 3.4.

As we did earlier, we first consider languages which have a natural bound on the number of shuffle productions that can occur in a derivation tree of any word in the language. That is followed by an example of a recursive shuffle grammar to illustrate how powerful they can be. It has been proven [17] that the recursive shuffle grammars do indeed have a greater expressive power, but it is not always clear how to interpret the resulting combinatorial families. We begin with a second kind of pointing operator.

3.1. A terminal pointing operator

The traditional pointing operator can be used to model $z\frac{d}{dz}$, but one can show that this is, in fact, insufficient to generate all D-finite functions. To remedy this, we define a pointing operator which mimics the concept behind the derivative of a species. This pointing operator has the effect of converting a letter to an epsilon by 'marking' the letter. Consequently, a letter can not be marked more than once, and each subsequent time a word is marked, there is a counter on the mark which is augmented. The pointing operator applied a set of words will be the pointing operator applied to each of the elements of the set. Notationally, we distinguish them with accumulated primes. We give some examples:

$$\begin{aligned}\mathcal{P}(aab) &= a'ab + aa'b + aab'\\ \mathcal{P}(\mathcal{P}(aab)) &= a'a''b + a'ab'' + a''a'b + aa'b'' + a''ab' + aa''b'\\ \mathcal{P}(a'''a'b'') &= \emptyset.\end{aligned}$$

The length of the word is the number of unmarked letters in a word (but the combinatorial objects in the language encode more than just the length in some sense). The number of words in the pointing of a word is equal to its length.

This gives a straightforward interpretation of the derivative:

$$\mathcal{L}_1 = \mathcal{P}(\mathcal{L}) \qquad \Longrightarrow \qquad L_1(z) = \frac{d}{dz}L(z).$$

Using this definition if A is a symbol which 'yields' through a grammar a language

Remark, if we allow concatenation after marking, we could generate two letters in the same word marked with a single prime via concatenation of marked words.

Using the marking operation, we can express most D-finite functions, specifically, by the differential equations that they satisfy. For example, the series $P(z) = \sum_{n\geq 0} n!z^n$ satisfies the differential equation

$$P(z) = 1 + zP(z) + z^2P'(z).$$

This is modelled by the grammar

$$A \to \varepsilon$$
$$A \to aA$$
$$A \to bc\mathcal{P}(A).$$

An alphabet on three letters (a, b, c) allows us to track the origin of each letter. Here is the result of the third iteration of the rules:

$$1 \oplus a \oplus aa + ba'c \oplus aaa + abca' + bca'a + bcb''ca' + bcaa' + bcbc''a' \oplus aaaa + aabca' + abca'a.$$

We will call a pointing grammar one that has rules of the form

$$A \to w, \qquad A \to wB, \qquad A \to \mathcal{P}(B). \tag{3.1}$$

Despite the fact that we allow only *left* concatenation, (a strategy to avoid concatenating pointed words) these grammars rules can model any D-finite function.

We can define a procedure for finding a language given a defining equation satisfied by a D-finite generating function. Say that a generating function $T(z)$ satisfies

$$T(z) = q(z) + q_0(z)T(z) + q_1(z)T'(z) + \ldots + q_n(z)T^{(n)}(z)\ . \tag{3.2}$$

Now substitute $T(z) = P(z) - N(z)$ and

$$(P(z)-N(z)) = q(z)+q_0(z)(P(z)-N(z))+q_1(z)(P'(z)-N'(z))+\ldots+q_n(z)(P^{(n)}(z)-N^{(n)}(z))$$

Use also the notation that $q_i(z) = q_i^+(z) - q_i^-(z)$ where $q_i^+(z)$ are the positive terms of the polynomial and $q_i^-(z)$ are the negative ones.

Then if

$$P(z) = q^+(z) + q_0^+(z)P(z) + q_0^-(z)N(z) + \cdots + q_n^+(z)P^{(n)}(z) + q_n^-(z)N^{(n)}(z) \tag{3.3}$$

and

$$N(z) = q^-(z) + q_0^-(z)P(z) + q_0^+(z)N(z) + \cdots + q_n^-(z)P^{(n)}(z) + q_n^+(z)N^{(n)}(z) \tag{3.4}$$

then $P(z) - N(z)$ satisfies equation (3.2).

Now we can define a language with a rule for each monomial in (3.3) and (3.4) and every terms $x^a R^{(k)}(z)$ is represented by a rule of the form

$$\tilde{R} \to w\mathcal{P}(\cdots \mathcal{P}(R)\cdots)$$

where $\mathcal{P}$ occurs k times and R, $\tilde{R}$ are symbols representing a language whose generating function is either $P(z)$ or $N(z)$ and w is a word of length a.

Any language which is generated from rules of the form Eq. (3.1) has a generating function which satisfies a linear differential equation, and hence is D-finite.

We summarize this in the following theorem.

Theorem 3.1. *A language which is generated from the rules of the form Eq.* (3.1) *has a D-finite generating function. Moreover, any D-finite function can be written as a difference of two generating functions for languages which are generated by rules of this form.*

3.2. Acyclic shuffle dependencies

We consider languages generated by the following re-writing rules, where w is a word, and A, B and C are non-terminals:

$$A \to w, \qquad A \to BC, \qquad A \to B \sqcup\!\sqcup C. \tag{3.5}$$

For any language generated by rules of the above type, and a fixed set of non-terminals, we construct the graph with non-terminals as nodes, and for every production rule $A \to B \sqcup\!\sqcup C$, we make an edge from A to B and an edge from A to C. If this graph is acyclic, we say the language has acyclic shuffle dependencies. The next section treats languages that have a cyclic dependency.

We prove that this class of languages is larger than those generated by the pointing operator of the previous section, because we can generate a language with a generating function that is not D-finite.

We re-use the Dyck languages $\mathcal{D}$ and $\mathcal{E}$ defined in Section 3.4. Consider the language generated by the following grammar:

$$A \to \mathcal{D} \sqcup\!\sqcup \mathcal{E}$$
$$C \to 1|AC.$$

The shuffle dependency graph is a tree, and thus this is in our class. The generating functions of A and C are given by

$$A(z) = \frac{-1}{4z} + \frac{(16z-1)}{2\pi z}\,\mathrm{EllipticK}(4\sqrt{z}) + \frac{1}{\pi z}\,\mathrm{EllipticE}(4\sqrt{z}), \quad C(z) = \frac{1}{1-A(z)}.$$

Since $1-A(z)$ is not of the form $\exp(algebraic)algebraic$, $C(z)$ is not D-finite. Nonetheless, we can prove an asymptotic result about generating functions in this class.

Theorem 3.2. *Let L be a proper language generated by shuffle production in an unambiguous grammar of with rules of the form given in Eq.* (3.5)*, on an alphabet with k letters. The number of words of length n, $\ell(n)$, satisfies $\ell(n) = O(n!^2)$.*

Proof. Since the grammar generates proper languages, there are no shuffle productions with epsilon. Thus, the derivation tree of a word of length n can have at most n shuffle productions. In the worst case, each one increments the alphabet and so the maximum size of alphabet that a word of length n can draw on is then kn. The total number of words from this alphabet is $(kn)^n$.

For $k < n$ the result follows by Stirling's formula. ∎

3.3. Cyclic shuffle dependencies

Languages in this class will have an infinite alphabet since we use a disjoint union in our shuffle. However, the number of words of a given length is finite if there is no derivation tree possible that is a shuffle and an ϵ. Under this restriction, any word of length n comes from an alphabet using no more than more than a constant multiple of n letters. We consider an important class of this type in the next section.

3.4. The shuffle closure of a languages

A class of languages which falls under this category are those that are generating using the shuffle closure operator. The *shuffle closure* of a language is defined recursively in the following way: $\mathcal{L}^{\sqcup\!\sqcup 1} = \mathcal{L} \sqcup\!\sqcup \mathcal{L}$, and $\mathcal{L}^{\sqcup\!\sqcup n} = \mathcal{L}^{\sqcup\!\sqcup n-1} \sqcup\!\sqcup \mathcal{L}$. The shuffle closure, is the union over all finite shuffles:

$$\mathcal{L}^{\sqcup\!\sqcup} = \bigcup_n \mathcal{L}^{\sqcup\!\sqcup n}.$$

Equivalently, we write this as a grammar production: $A \to A \sqcup\!\sqcup B | B$. The shuffle closure [16, 17] provides extremely concise notation. In particular, they arise in descriptions of sequential execution histories of concurrent processes.

Remark, that the closure of the language is one single language, whereas the closure of the class of languages that is one language is an infinite set of languages.

The shuffle closure of a single letter gives all permutations:

$$a^{\sqcup\!\sqcup} = a \oplus \overline{a}a + a\overline{a} \oplus \overline{\overline{a}}\overline{a}a + \overline{\overline{a}}a\overline{a} + a\overline{\overline{a}}\overline{a} + \overline{\overline{a}}\overline{a}a + \overline{a}a\overline{\overline{a}} + a\overline{\overline{a}}\overline{\overline{a}} \oplus \dots$$

The generating function of the this language is $\sum n! z^n$, and indeed the generating function of the shuffle closure of any word of length k is $\sum (kn)!(\frac{z^k}{k})^n$, which is also D-finite.

To prove our formula above, we express the generating function of $\mathcal{L}^{\sqcup\!\sqcup}$ in terms of the operators which switch between the ordinary and exponential generating functions. Recall, $L(z) = \sum a_n z^n \implies \hat{L}(z) = \sum \frac{a_n}{n!}$, and we define the Laplace operator $\mathcal{L} \cdot \hat{L}(z) = L(z)$. Then,

$$\mathcal{L}_1 = \mathcal{L}^{\sqcup\!\sqcup} \qquad \Longrightarrow \qquad L_1(z) = \sum_n \mathcal{L} \cdot [(\hat{L}(z))^n]. \tag{3.6}$$

Although all of the summands are D-finite, it is possible that the sum is not.

Clearly, the shuffle closure does not preserve regularity, and indeed adding it, and the shuffle product to regular languages is enough to generate all recursively enumerable languages. Thus, we see that if there is no bound on the number of shuffles possible in any expression tree, the languages can get far more complex.

Nonetheless the following conjecture seems reasonable, and perhaps it is possible to prove it following starting from Eq. (3.6), and necessarily a more sophisticated analysis.

Conjecture 3.3. The shuffle closure of a regular language has a D-finite generating function.

4. Conclusion

A next step is to adapt the Bolzmann generators to these languages. Since we can effectively simulate labelled objects in an unlabelled context, we can easily describe objects like strong interval trees. This approach might allow a detailed analysis of certain parameters of permutation sorting by reversals, as applied to comparative genomics [1].

We are also interested in characterizing the context-free languages whose shuffle is not algebraic, and to consider the other naturual questions of closure that are standard for language classes.

Acknowledgments. We gratefully acknowledge many discussions from the Algebraic Combinatorics Seminar at the Fields Institute. In particular, we acknowledge contributions by N. Bergeron, C. Hollweg, and M. Rosas. We wish to also acknowledge the financial support of NSERC.

References

[1] Séverine Bérard, Anne Bergeron, Cedric Chauve, and Chistophe Paul. Perfect sorting by reversals is not always difficult. *IEEE/ACM Trans. on comput. biology and bioinformatics*, 4(1), 2007.

[2] F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial species and tree-like structures*, volume 67 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1998.

[3] François Bergeron and Christophe Reutenauer. Combinatorial resolution of systems of differential equations. III. A special class of differentially algebraic series. *European J. Combin.*, 11(6):501–512, 1990.

[4] François Bergeron and Ulrike Sattler. Constructible differentially finite algebraic series in several variables. *Theoret. Comput. Sci.*, 144(1-2):59–65, 1995.

[5] Manuel Bodirsky, Éric Fusy, Mihyun Kang, and Stefan Vigerske. An unbiased pointing operator for unlabeled structures, with applications to counting and sampling. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *SODA*, pages 356–365. SIAM, 2007.

[6] Mireille Bousquet-Mélou. Algebraic generating functions in enumerative combinatorics, and context-free languages. In *Stacs 05*, volume 3404 of *Lecture Notes in Comput. Sci.*, pages 18–35. Springer, 2005.

[7] Mireille Bousquet-Mélou. Rational and algebraic series in combinatorial enumeration. In *International Congress of Mathematicians*, pages 789–826, 2006.

[8] Frédéric Chyzak, Marni Mishna, and Bruno Salvy. Effective scalar products of D-finite symmetric functions. *J. Combin. Theory Ser. A*, 112(1):1–43, 2005.

[9] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combin. Probab. Comput.*, 13(4-5):577–625, 2004.

[10] Philippe Flajolet. Analytic models and ambiguity of context-free languages. *Theoret. Comput. Sci.*, 49(2-3):283–309, 1987.

[11] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Appl. Math.*, 39(3):207–229, 1992.

[12] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. `http://algo.inria.fr/flajolet/Publications/books.html`, 2006.

[13] Philippe Flajolet, Paul Zimmerman, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoret. Comput. Sci.*, 132(1-2):1–35, 1994.

[14] Jay Gischer. Shuffle languages, petri nets, and context-sensitive grammars. *Communications of the ACM*, 24(9), September 1981.

[15] Daniel Hill Greene. *Labelled Formal Languages and Their Uses*. PhD thesis, Stanford University, 1983.

[16] Matthias Jantzen. Extending regular expressions with iterated shuffle. *Theoret. Comput. Sci.*, 38(2-3):223–247, 1985.

[17] Joanna Jędrzejowicz. Infinite hierarchy of expressions containing shuffle closure operator. *Inform. Process. Lett.*, 28(1):33–37, 1988.

[18] Gilbert Labelle and Cédric Lamathe. A theory of general combinatorial differential operators. In *Formal Power Series and Algebraic Combinatorics*, 2007.

[19] L. Lipshitz. The diagonal of a D-finite power series is D-finite. *J. Algebra*, 113(2):373–378, 1988.

[20] M. Lothaire. *Combinatorics on words*, volume 17 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley Publishing Co., Reading, Mass., 1983.

[21] Marni Mishna. Automatic enumeration of regular objects. *J. Integer Sequences*, 10:Article 07.5.5, 2007.

[22] Lee A. Rubel. A survey of transcendentally transcendental functions. *Amer. Math. Monthly*, 96(9):777–788, 1989.

[23] Michael F. Singer. Algebraic relations among solutions of linear differential equations. *Trans. Amer. Math. Soc.*, 295(2):753–763, 1986.

[24] Richard P. Stanley. *Enumerative combinatorics. Vol. 2*, volume 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1999.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 573-584
www.stacs-conf.org

WEAK INDEX VERSUS BOREL RANK

FILIP MURLAK [1]

[1] Warsaw University
E-mail address: fmurlak@mimuw.edu.pl

Abstract. We investigate weak recognizability of deterministic languages of infinite trees. We prove that for deterministic languages the Borel hierarchy and the weak index hierarchy coincide. Furthermore, we propose a procedure computing for a deterministic automaton an equivalent minimal index weak automaton with a quadratic number of states. The algorithm works within the time of solving the emptiness problem.

1. Introduction

Finite automata on infinite trees are one of the basic tools in the verification of nonterminating programs. Practical applicability of this approach relies on the simplicity of the automata used to express the specifications. On the other hand it is convenient to write the specifications in an expressive language, e. g. μ-calculus. This motivates the search for automatic simplifications of automata. An efficient, yet reasonably expressive, model is offered by weak alternating automata. It was essentially showed by Rabin [18] that a language L can be recognized by a weak automaton if and only if both L and $L^{\complement}$ can be recognized by nondeterministic Büchi automata. Arnold and Niwiński [2] proposed an algorithm that, given two Büchi automata recognizing a language and its complement, constructs a doubly exponential alternation free μ-calculus formula defining L, which essentially provides an equally effective translation to a weak automaton. Kupferman and Vardi [7] gave an immensely improved construction that involves only quadratic blow-up.

A more refined construction could also simplify an automaton in terms of different complexity measures. A measure that is particularly important for theoretical and practical reasons is the Mostowski–Rabin index. This measure reflects the alternation depth of positive and negative events in the behaviour of a verified system. The index orders automata into a hierarchy that was proved strict for deterministic [21], nondeterministic [13], alternating [4, 8], and weak alternating automata [9]. Computing the least possible index for a given automaton is called the index problem. Unlike for ω-words, where the solution was essentially given already by Wagner [21], for trees this problem in its general form

1998 ACM Subject Classification: F.1.1, F.4.1, F.4.3.
Key words and phrases: weak index, Borel rank, deterministic tree automata.
Supported by the Polish government grant no. N206 008 32/0810.

remains unsolved. For deterministic languages, Niwiński and Walukiewicz gave algorithms to compute the deterministic and nondeterministic indices [14, 16].

The theoretical significance of the weak index is best reflected by its coincidence with the quantifier alternation depth in the weak monadic second order logic [9]. Further interesting facts are revealed by the comparison with the Borel rank. In 1993 Skurczyński gave examples of Π^0_n and Σ^0_n-complete languages recognized by weak alternating automata with index $(0, n)$ and $(1, n+1)$ accordingly [19]. In [5] it was shown that weak $(0, n)$-automata can only recognize Π^0_n languages (and dually, $(1, n+1)$-automata can only recognize Σ^0_n languages), and it was conjectured that the weak index and the Borel hierarchies actually coincide. Here we prove that the conjecture holds for deterministic languages. Consequently, the algorithm calculating the Borel rank for deterministic languages [11] can be also used to compute the weak index. Since all deterministic languages are at the first level of the alternating hierarchy, this completes the picture for the deterministic case. We also provide an effective translation to a weak automaton with a quadratic number of states and the minimal index.

2. Automata

We will be working with deterministic and weak automata, but to have a uniform framework, we first define automata in their most general alternating form.

A *parity game* is a perfect information game of possibly infinite duration played by two players, Adam and Eve. We present it as a tuple $(V_\exists, V_\forall, E, v_0, \mathrm{rank})$, where $V_\exists$ and $V_\forall$ are (disjoint) sets of positions of Eve and Adam, respectively, $E \subseteq V \times V$ is the relation of possible moves, with $V = V_\exists \cup V_\forall$, $p_0 \in V$ is a designated initial position, and $\mathrm{rank} : V \to \{0, 1, \ldots, n\}$ is the ranking function.

The players start a play in the position v_0 and then move a token according to relation E (always to a successor of the current position), thus forming a path in the graph (V, E). The move is selected by Eve or Adam, depending on who is the owner of the current position. If a player cannot move, she/he looses. Otherwise, the result of the play is an infinite path in the graph, $v_0, v_1, v_2, \ldots$. Eve wins the play if the highest rank visited infinitely often is even, otherwise Adam wins.

An *alternating automaton* $A = \langle \Sigma, Q_\exists, Q_\forall, q_0, \delta, \mathrm{rank}\rangle$, consists of a finite input alphabet Σ, a finite set of states Q partitioned into existential states $Q_\exists$ and universal states $Q_\forall$ with a fixed initial state q_0, a transition relation $\delta \subseteq Q \times \Sigma \times \{0, 1, \varepsilon\} \times Q$, and a ranking function $\mathrm{rank} : Q \to \omega$. Instead of $(p, \sigma, d, q) \in \delta$, one usually writes $p \xrightarrow{\sigma,d} q$.

An input tree t is accepted by A iff Eve has a winning strategy in the parity game $\langle Q_\exists \times \{0,1\}^*, Q_\forall \times \{0,1\}^*, (q_0, \varepsilon), E, \mathrm{rank}'\rangle$, where $E = \{((p, v), (q, vd))\colon v \in \mathrm{dom}(t),\ (p, t(v), d, q) \in \delta\}$ and $\mathrm{rank}'(q, v) = \mathrm{rank}(q)$. The computation tree of A on t is obtained by unravelling the graph above from the vertex (q_0, ε) and labelling the node $(q_0, \varepsilon), (q_1, d_1), (q_2, d_2), \ldots, (q_n, d_n)$ with q_n. The result of the parity game above only depends on the computation tree.

An automaton is called *deterministic* iff Eve has no choice at all, and Adam can only choose the direction: left or right (no ε-moves). Formally, it means that $Q_\exists = \emptyset$, and $\delta : Q \times \Sigma \times \{0, 1\} \to Q$. For deterministic automata, the computation tree is a full binary tree. The transitions are often written as $p \xrightarrow{\sigma} q_0, q_1$, meaning $p \xrightarrow{\sigma,d} q_d$ for $d = 0, 1$.

A *weak automaton* is an alternating automaton satisfying the condition

$$p \xrightarrow{\sigma,d} q \quad \Longrightarrow \quad \mathrm{rank}\, p \leq \mathrm{rank}\, q\,.$$

A more elegant definition of the class of weakly recognizable languages is obtained by using *weak parity games* in the definition of acceptance by alternating automata. In those games Eve wins a play if the highest rank used at least once is even. For the purpose of the following lemma, let us call the first version *restricted alternating automata*. Later, we will stick to the second definition.

Lemma 2.1. *For every L it holds that L is recognized by a restricted alternating (ι, κ)-automaton iff it is recognized by a weak alternating (ι, κ)-automaton.*

Proof. Every restricted automaton can be transformed into an equivalent weak automaton by simply changing the acceptance condition to weak. Let us, then, concentrate on the converse implication.

Fix a weak automaton A using ranks (ι, κ). To construct a restricted automaton we will take one copy of A for each rank: $A^{(\iota)}, A^{(\iota+1)}, \ldots, A^{(\kappa)}$. By $q^{(i)}$ we will denote the counterpart of A's state q in $A^{(i)}$. We set $\operatorname{rank} q^{(i)} = i$. We want the number of the copy the computation is in to reflect the highest rank seen so far. To obtain that, we set the initial state of the new automaton to $q_0^{(\operatorname{rank} q_0)}$, and for each i and each transition $p \xrightarrow{\sigma,d} q$ in A we add a transition $p^{(i)} \xrightarrow{\sigma,d} q^{(\max(i, \operatorname{rank} q))}$. For each i and q, $q^{(i)}$ is universal iff q is universal. Checking the equivalence is straightforward. ∎

For deterministic automata we will assume that all states are productive, i. e., are used in some accepting run, save for one all-rejecting state $\bot$, and that all transitions are productive or go to $\bot$, i. e., whenever $q \xrightarrow{\sigma} q_1, q_2$, then either q_1 and q_2 are productive, or $q_1 = q_2 = \bot$. The assumption of productivity is vital for our proofs. Thanks to this assumption, in each node of an automaton's run we can plug in an accepting sub-run.

Transforming a given automaton into such a form of course needs calculating the productive states, which is equivalent to deciding a language's emptiness. The latter problem is known to be in NP ∩ co-NP, but it has no polynomial solutions yet. Therefore we can only claim that our algorithms are polynomial for the automata that underwent the above preprocessing. We will try to mention it whenever particularly important.

3. Two Hierarchies

The *index of an automaton* A is a pair $(\min \operatorname{rank} Q, \max \operatorname{rank} Q)$. Scaling down the rank function if necessary, one may assume that $\min \operatorname{rank} Q$ is either 0 or 1. Thus, the indices are elements of $\{0,1\} \times \omega \setminus \{(1,0)\}$. For an index (ι, κ) we shall denote by $\overline{(\iota, \kappa)}$ *the dual index*, i. e., $\overline{(0,\kappa)} = (1, \kappa+1)$, $\overline{(1,\kappa)} = (0, \kappa-1)$. Let us define an ordering of indices with the following formula:

$$(\iota, \kappa) < (\iota', \kappa') \text{ if and only if } \kappa - \iota < \kappa' < \iota' .$$

In other words, one index is greater than another if and only if it "uses" more ranks. This means that dual indices are incomparable. *The Mostowski–Rabin index hierarchy* for a certain class of automata consists of ascending sets (levels) of languages recognized by (ι, κ)-automata.

Here, we are mainly interested in the *weak index hierarchy*, i. e., the hierarchy of languages recognized by weak (ι, κ)-automata. The strictness of this hierarchy was established by Mostowski [9] via equivalence with the quantifier-alternation hierarchy for the weak

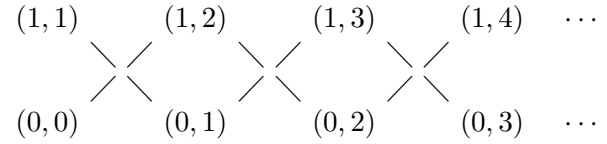


Figure 1: The Mostowski–Rabin index hierarchy

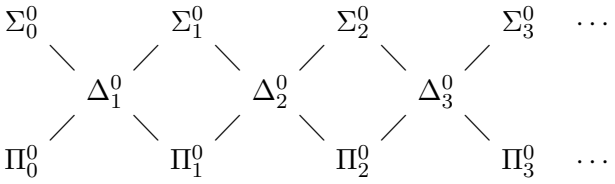


Figure 2: The Borel hierarchy

monadic second order logic, whose strictness was proved by Thomas [20]. The *weak index problem*, i. e., computing the minimal weak index needed to recognize a given weak language, for the time being remains unsolved just like other versions of the index problem.

The weak index hierarchy is closely related to the Borel hierarchy. We will work with the standard Cantor-like topology on T_Σ induced by the metric

$$d(s,t) = \begin{cases} 2^{-\min\{|x|\,:\, x\in\{0,1\}^*,\, s(x)\neq t(x)\}} & \text{iff } s \neq t \\ 0 & \text{iff } s = t \end{cases}.$$

The class of Borel sets of a topological space X is the closure of the class of open sets of X by countable sums and complementation.

For a topological space X, the initial (finite) levels of the *Borel hierarchy* are defined as follows:

- $\Sigma^0_1(X)$ – open subsets of X,
- $\Pi^0_k(X)$ – complements of the sets from $\Sigma^0_k(X)$,
- $\Sigma^0_{k+1}(X)$ – countable unions of sets from $\Pi^0_k(X)$.

For instance, $\Pi^0_1(X)$ are the closed sets, $\Sigma^0_2(X)$ are F_σ sets and $\Pi^0_2(X)$ are G_δ sets. By convention $\Sigma^0_0(X) = \{\emptyset\}$ and $\Pi^0_0(X) = \{X\}$.

A straightforward inductive argument shows that the classes defined above are closed under inverse images of continuous functions. Let $\mathcal{C}$ be one of those classes. A set A is called *$\mathcal{C}$-hard*, if each set in $\mathcal{C}$ is an inverse image of A under some continuous function. If additionally $A \in \mathcal{C}$, A is *$\mathcal{C}$-complete*.

We start the discussion of the relations between the index of a weak automaton and the Borel rank of the language it recognises by recalling Skurczyński's results. For a tree $t : \{0,1\}^* \to \Sigma$ and a node $v \in \{0,1\}^*$ let $t.v$ denote the tree rooted in v, i. e., $t.v(w) = t(vw)$. Let us define a sequence of languages:

- $L_{(0,1)} = \{t\}$, where $t \in T_{\{a,b\}}$ is the tree with no b's,
- $L_{(1,n+1)} = L^{\complement}_{(0,n)}$ for $n \geq 1$,
- $L_{(0,n+1)} = \{t \in T_{\{a,b\}} :\ \forall_k\, t.0^k1 \in L_{(1,n+1)}\}$ for $n \geq 1$.

Theorem 3.1 (Skurczyński [19])**.** *For each $n \geq 1$,*

- *$L_{(0,n)}$ is a Π^0_n-complete language recognized by a weak $(0,n)$-automaton,*

- *$L_{(1,n+1)}$ is a Σ^0_n-complete language recognized by a weak $(1,n+1)$-automaton.*

We will now show that this construction is as efficient as it can be: ranks $(0,n)$ are necessary to recognize any Π^0_n-hard language (if it can be weakly recognized at all).

We will actually prove a bit stronger result. We will consider *weak game languages* $W_{[\iota,\kappa]}$, to which all languages recognized by weak $[\iota,\kappa]$-automata can be reduced, and show that $W_{[0,n]} \in \Pi^0_n$ and $W_{[1,n+1]} \in \Sigma^0_n$ (by Skurczyński's results, they are hard for these classes). The languages $W_{[\iota,\kappa]}$ are natural weak counterparts of strong game languages that prove the strictness of the strong alternating index hierarchy. Lately Arnold and Niwiński proved that the strong game languages also form a strict hierarchy with respect to continuous reductions, but they are all non-Borel [3].

Fix a natural number N. For $\iota = 0, 1$ and $\kappa \geq \iota$, let $\mathcal{T}_{(\iota,\kappa)}$ denote the set of full N-ary trees over the alphabet $\{\exists,\forall\} \times \{\iota, \iota+1, \ldots, \kappa\}$. Let $W_{(\iota,\kappa)} \subseteq \mathcal{T}_{(\iota,\kappa)}$ be the set of all trees t for which Eve has a winning strategy in the *weak* parity game $G_t = \langle V_\exists, V_\forall, E, v_0, \mathrm{rank}\rangle$, where $V_\theta = \{v \in \mathrm{dom}\, t\colon t(v) = (\theta, j) \text{ for some } j\}$, $E = \{(v, vk)\colon v \in \mathrm{dom}\, t,\ k < N\}$, $v_0 = \varepsilon$, $\mathrm{rank}(v) = j$ iff $t(v) = (\theta, j)$ for some θ.

Theorem 3.2. *For each n, $W_{(0,n)} \in \Pi^0_n(\mathcal{T}_{(0,n)})$ and $W_{(1,n+1)} \in \Sigma^0_n(\mathcal{T}_{(1,n+1)})$.*

Proof. We will proceed by induction on n. For $n = 0$ the claim is obvious: $W_{(0,0)} = \mathcal{T}_{(0,0)} \in \Pi^0_0(\mathcal{T}_{(0,0)})$, $W_{(1,1)} = \emptyset \in \Sigma^0_0(\mathcal{T}_{(1,1)})$.

Take $n > 0$. For each $t \in W_{(1,n+1)}$ there exists a strategy σ for Eve, such that it guarantees that the play reaches a node with the rank greater or equal to 2. By König lemma, this must happen in a bounded number of moves. Basing on this observation we will provide a Σ^0_n presentation of $W_{(1,n+1)}$.

Let *k-antichain* be a subset of the nodes on the level k. Let $\mathcal{A}$ denote the set of all possible k-antichains for all $k < \omega$. Obviously this set is countable. For a k-antichain A let W_A denote the set of trees such that there exists a strategy for Eve that guarantees visiting a node with the rank ≥ 2 during the initial k moves and reaching a node from A. This set is a clopen. We have a presentation

$$W_{(1,n+1)} = \bigcup_{A\in\mathcal{A}} \left(W_A \cap \bigcap_{v\in A} \left\{ t : t'.v \in W_{(0,n-1)} \right\} \right),$$

where t' is obtained from t by decreasing all the ranks by 2 (if the result is -1, take 0). The claim follows by induction hypothesis and the continuity of $t \mapsto t'$ and $t \mapsto t.v$.

Now, it remains to see that $W_{(0,n)} \in \Pi^0_n(\mathcal{T}_{(0,n)})$. For this, note that

$$W_{(0,n)} = \left\{ t : t'' \in (W_{(1,n+1)})^{\complement} \right\},$$

where t'' is obtained from t by swapping $\exists$ and $\forall$, and increasing ranks by 1. The claim follows by the continuity of $t \mapsto t''$. ∎

As a corollary we get the promised improvement of Skurczyński's result.

Corollary 3.3. *For every weak alternating automaton A with index $(0,n)$ (resp. $(1,n+1)$) it holds that $L(A) \in \Pi^0_n$ (resp. $L(A) \in \Sigma^0_n$).*

Proof. Let A be an automaton with priorities inside $[\iota,\kappa]$. For sufficiently large N we may assume without loss of generality that the computation trees of the automaton are N-ary trees. By assigning to an input tree the run of A, one obtains a continuous function reducing $L(A)$ to $W_{(\iota,\kappa)}$. Hence, the claim follows from the theorem above. ∎

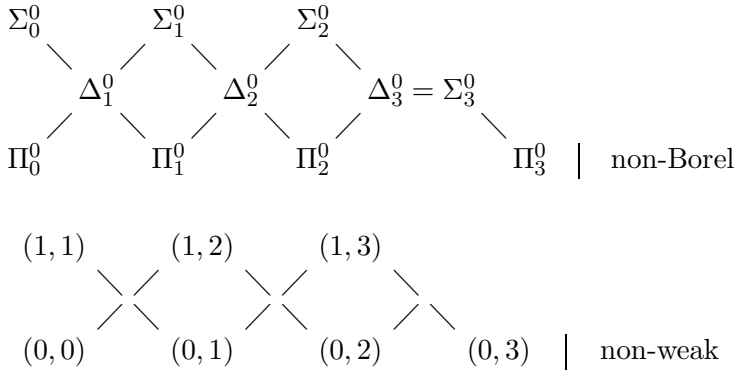


Figure 3: The Borel hierarchy and weak index hierarchy for deterministic tree languages.

In fact the corollary follows also from Mostowski's theorem on equivalence of weak automata and weak monadic second order logic on trees [9]. The present proof of Theorem 3.2 is actually just a repetition of Mostowski's proof in the setting of the Borel hierarchy. An entirely different proof can be found in [5].

We believe that the converse implication is also true: a weakly recognizable Π^0_n-language can be recognized by a weak $(0,n)$-automaton (and dually for Σ^0_n).

Conjecture 3.4. For weakly recognizable languages the weak index hierarchy and the Borel hierarchy coincide.

In this paper we show that the conjecture holds true when restricted to deterministic languages.

4. The Deterministic Case

In 2002 Niwiński and Walukiewicz discovered a surprising dichotomy in the family of deterministic languages: a deterministic language is either very simple or very sophisticated.

Theorem 4.1 (Niwiński, Walukiewicz [15])**.** *For a deterministic automaton A with n states, $L(A)$ is either recognizable with a weak alternating $(0,3)$-automaton with $\mathcal{O}(n^2)$ states (and so Π^0_3) or is non-Borel (and so not weakly recognizable). The equivalent weak automaton can be constructed within the time of solving the emptiness problem.*

An important tool used in the proof of the Gap Theorem (Theorem 4.1) is the technique of difficult patterns. In the topological setting the general recipe goes like this: for a given class identify a pattern that can be unravelled to a language complete for this class; if an automaton does not contain the pattern, then $L(A)$ should be in the dual class. The same technique was later applied to obtain effective characterisations of the remaining Borel classes of deterministic languages [11].

Let us define the patterns used in these characterisations. A *loop* in an automaton is a sequence of states and transitions:

$$p_0 \xrightarrow{\sigma_1,d_1} p_1 \xrightarrow{\sigma_2,d_2} \dots \xrightarrow{\sigma_n,d_n} p_0\,.$$

A loop is called *accepting* if $\max_i \operatorname{rank}(p_i)$ is even. Otherwise it is *rejecting*.

A (ι, κ)-*flower* is a sequence of loops $\lambda_\iota, \lambda_{\iota+1}, \ldots, \lambda_\kappa$ starting in the same state p, such that the highest rank appearing on λ_i has the same parity as i and it is higher than the highest rank on λ_{i-1} for $i = \iota, \iota+1, \ldots, \kappa$.

A *weak* (ι, κ)-*flower* is a sequence of loops $\lambda_\iota, \lambda_{\iota+1} \ldots, \lambda_\kappa$ such that λ_{i+1} is reachable from λ_i, and λ_i is accepting iff i is even.

A *split* is a pair of loops $p \xrightarrow{\sigma,0} p_0 \longrightarrow \ldots \longrightarrow p$ and $p \xrightarrow{\sigma,1} p_1 \longrightarrow \ldots \longrightarrow p$ such that the highest ranks occurring on them are of different parity and the highest one is odd.

A state q is *replicated* by a loop $p \xrightarrow{\sigma,d_0} p_0 \longrightarrow \ldots \longrightarrow p$ if there exists a path $p \xrightarrow{\sigma,d_1} p_1 \longrightarrow \ldots \longrightarrow q$ such that $d_0 \neq d_1$. We will say that a loop or a flower is replicated by a loop λ if it contains a state replicated by λ.

Proposition 4.2 (Niwiński, Walukiewicz [15]; Murlak [11])**.** *Let A be a deterministic automaton.*

(1) $L(A) \in \Pi^0_1$ *iff A contains no weak* $(1,2)$*-flower.*
(2) $L(A) \in \Sigma^0_1$ *iff A contains no weak* $(0,1)$*-flower.*
(3) $L(A) \in \Pi^0_2$ *iff A contains no* $(0,1)$*-flower.*
(4) $L(A) \in \Sigma^0_2$ *iff A contains neither* $(1,2)$*-flower nor a weak* $(1,2)$*-flower replicated by an accepting loop.*
(5) $L(A) \in \Sigma^0_3$ *iff A contains no* $(0,1)$*-flower replicated by an accepting loop.*
(6) $L(A) \in \Pi^0_3$ *iff A contains no split.*

In particular, the Borel rank of $L(A)$ is computable within the time of finding the productive states of A.

The patterns defined above were originally introduced to capture the index complexity of recognizable languages. Niwiński and Walukiewicz used flowers to solve the deterministic index problem for word languages [14]. Their result may easily be adapted to trees (see [11] for details).

Theorem 4.3. *For a deterministic tree automaton A the language $L(A)$ is recognized by a deterministic (ι, κ)-automaton iff A does not contain a $\overline{(\iota, \kappa)}$-flower. An equivalent minimal index automaton with the same number of states can be constructed within the time of solving the emptiness problem.*

The weak flowers provide an analogous characterisation of the weak deterministic index.

Proposition 4.4 ([11])**.** *A deterministic automaton A is equivalent to a weak deterministic (ι, κ)-automaton iff it does not contain a weak $\overline{(\iota, \kappa)}$-flower. An equivalent minimal index automaton with the same number of states can be constructed within the time of solving the emptiness problem.*

Proof. If the automaton contains a weak (ι, κ)-flower, for each weak $\overline{(\iota, \kappa)}$-automaton one can build a cheating tree (see [11] for details). For the converse implication, construct a weak deterministic (ι, κ)-automaton by modifying the ranks of the given deterministic automaton. Set $\operatorname{rank} q$ to the lowest number m such that there exists a weak (m, κ)-flower with a path from q to λ_m. ∎

5. The Power of the Weak

In this section we finally turn to the weak recognizability of deterministic languages. First we give sufficient conditions for a deterministic automaton to be equivalent to a weak alternating automaton of index $(0,2)$, $(1,3)$, and $(1,4)$. This is the first step to the solution of the weak index problem for deterministic automata.

Proposition 5.1. *For each deterministic $(1,2)$-automaton with n states one can construct an equivalent weak $(0,2)$-automaton with $2n+1$ states.*

Proof. Fix a deterministic $(1,2)$-automaton A. We will construct a weak $(0,2)$-automaton B such that $L(A) = L(B)$. Basically, for each node v the automaton B should check whether on each path in the subtree rooted in v the automaton A will reach a state with rank 2. This can be done as follows. Take two copies of A. In the first copy, all states are universal and have rank 0. The transitions are like in A plus for each state $q^{(1)}$ there is an ε-transition to $q^{(2)}$, the counterpart of $q^{(1)}$ in the second copy. In the second copy all states are universal and have rank 1. For the states with rank 1 in A, the transitions are like in A. For the states with rank 2 in A, there is just one transition to an all-accepting state $\top$ (rank 2 in B). ∎

Before we proceed with the conditions, let us show a useful property of the replication.

Lemma 5.2 (Replication Lemma)**.** *A state occurs in infinitely many incomparable nodes of an accepting run iff it is productive and is replicated by an accepting loop.*

Proof. If a state p is replicated by an accepting loop, then by productivity one may easily construct an accepting run with infinitely many incomparable occurrences of p. Let us concentrate on the converse implication.

Let p occur in an infinite number of incomparable nodes $v_0, v_1, \ldots$ of an accepting run ρ. Let π_i be a path of ρ going through the node v_i. Since 2^ω is compact, we may assume, passing to a subsequence, that the sequence π_i converges to a path π. Since v_i are incomparable, v_i is not on π. Let the word α_i be the sequence of states labeling the path from the last common node of π and π_i to v_i. Cutting the loops off if needed, we may assume that $|\alpha_i| \leq |Q|$ for all $i \in \omega$. Consequently, there exist a word α repeating infinitely often in the sequence $\alpha_0, \alpha_1, \ldots$. Moreover, the path π is accepting, so the starting state of α must lay on an accepting productive loop. This loop replicates p. ∎

Proposition 5.3. *For each deterministic $(0,1)$-automaton with n states which contains no weak $(1,2)$-flower replicated by an accepting loop one can construct effectively an equivalent weak $(1,3)$-automaton with $3n+1$ states.*

Proof. Let A be a deterministic $(0,1)$-automaton which contains no weak $(1,2)$-flower replicated by an accepting loop. Let us call a state of A *relevant* if it has the highest rank on some loop. We may change the ranks of productive irrelevant states to 0, and assume from now on that all odd states are relevant. We claim that the odd states occur only finitely many times on accepting runs of A. Suppose that an odd state p occurs infinitely many times in an accepting run ρ. Then it must occur in infinitely many incomparable nodes (otherwise we would get a rejecting path). By the Replication Lemma p is replicated by an accepting loop. As p is odd and relevant, it lies on some nontrivial rejecting loop. Since p is also productive, some accepting loop can be reached from p. Hence, A contains a weak $(1,2)$-flower replicated by an accepting loop - a contradiction

Now, we can easily construct a weak $(1,3)$-automaton recognising $L(A)$. Intuitively, we will simulate A and check if A's odd states occur finitely many times. This can be done as follows. Take three copies of A. In the first copy all the states are universal and have rank 1. The transitions are just like in A, only they go to the second copy of A. In the second copy of A, all the states are existential and have rank 1. From each state $q^{(2)}$ there are two ε-transitions to $q^{(1)}$ in the first copy and to $q^{(3)}$ in the third copy. Finally, in the third copy of A all the states are universal and have rank 2. The transitions from the states ranked 0 in A are just like in A, and from the states ranked 1 in A they go to an all-rejecting state $\bot$ (rank 3 in B). It is easy to see that B recognizes $L(A)$. ∎

Proposition 5.4. *For each automaton with n states containing no $(0,1)$-flower replicated by an accepting loop one can construct an equivalent weak alternating $(1,4)$-automaton with $\mathcal{O}(n^2)$ states.*

Proof. Let A be an automaton without $(0,1)$-flower replicated by an accepting loop. Consider the DAG of strongly connected components of A.For each SCC X containing at least one loop we will construct a weak automaton B_X recognising the languages of trees t such that each path of A's run on t that enters X either leaves X or is accepting. Obviously, the conjunction of such automata recognizes exactly $L(A)$. Let us first consider components replicated by an accepting loop. By the hypothesis, such a component must not contain a $(0,1)$-flower. Therefore we may assume that X only uses ranks 1 and 2. To obtain B_X take a copy of A. The states outside X can be divided into three disjoint groups: those that can be reached from X, those from which X can be reached, and the rest. Give the states from the first group the rank 4, and the states from the second and third group the rank 2. Finally, following the method from Proposition 5.1, replace X with an equivalent weak alternating subautomaton using ranks 2,3, and 4. The constructed automaton has $\mathcal{O}(n)$ states.

The case of X not replicated by an accepting loop is more tricky. The key property follows from the Replication Lemma. Let ρ_X denote the restriction of the run ρ to the nodes labeled with a state from X or having a descendant labeled with a state from X. By the Replication Lemma, this tree has only finitely many branches (some of them may be infinite). What B_X should do is to guess a node v on each path such that in the subtree rooted in v, ρ_X is either empty or consists of one infinite accepting branch. In the latter case we may additionally demand that on this infinite path the highest rank that ever occurs, occurs infinitely many times.

B_X consists of the component C_{guess} realising the guessing, the component $C_{A\setminus X}$ checking that no path of the computation enters X, and components $C_{X,r}$ for all ranks r used in X, which check that in a given subtree of the run ρ there is exactly one branch of ρ_X and that on this branch r occurs infinitely often and no higher rank is used.

To construct C_{guess}, take a copy of A and declare all the states universal and set their ranks to 1. For each q add a fresh existential state q' of rank 1 with an ε-transition to q and either to $q^{A\setminus X} \in C_{A\setminus X}$ if $q \notin X$ (ρ_X is empty) or to $q^{X,r} \in C_{X,r}$ for all r if $q \in X$ (ρ_X is one infinite accepting path). Finally replace each transition $p \xrightarrow{\sigma} p_0, p_1$ with $\xrightarrow{\sigma} p'_0, p'_1$.

The component $C_{A\setminus X}$ is a copy of A with all ranks equal 2, and the SCC X replaced with one all-rejecting state $\bot$ with rank 3.

Finally, let us now describe the automaton $C_{X,r}$. The automaton, staying in rank 2, works its way down the input tree just like A would, with the following modifications:

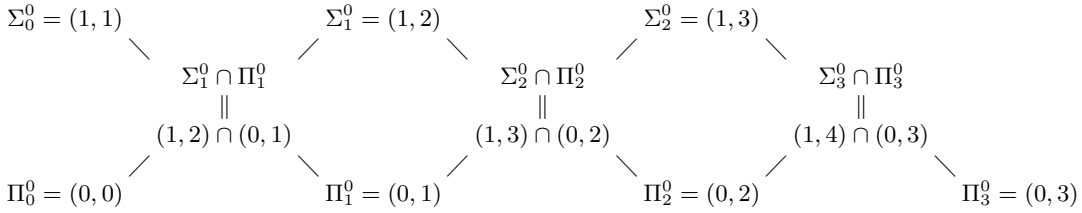


Figure 4: For deterministic tree languages the hierarchies coincide.

- if A enters a state in X with rank greater than r, $C_{X,r}$ moves to an all rejecting state $\perp$ (rank 3),
- if A takes a transition exiting X on both branches or staying in X on both branches, $C_{X,r}$ moves to $\perp$,
- if A takes a transition whose left branch leaves X and the right branch stays inside, $C_{X,r}$ sends to the right a $(3,4)$-component looking for a state from X with the rank r, and moves on to the right subtree (and symmetrically).

In order two see that $C_{X,r}$ does the job, it is enough to observe that if the $(3,4)$ component always succeeds to find a state from X with the rank r, then on the unique path that stays forever in X the rank r repeats infinitely often.

The $(3,4)$-component of $C_{X,r}$ can be constructed in such a way that it has $|X|+2$ states, and so in this case B_X has at most $2|X|(|X|+2)+3n \leq 2|X|^2+7n$ states.

In both cases, the number of states of B_X can be bounded by $c_1|X|^2 + c_2 n$ for fixed constants c_1 and c_2, independent of X. Since the SCCs are disjoint, the number of states of the conjunction of B_X's is at most

$$1 + \sum_{X \in A} (c_1|X|^2 + c_2 n) \leq 1 + c_1 \left(\sum_{X \in A} |X| \right)^2 + c_2 n^2 \leq (c_1 + c_2) n^2 + 1 \,.$$

■

We have now collected all the ingredients for the solution of the weak index problem for deterministic languages. What is left to be done is to glue together the sufficient conditions for index easiness and Borel hardness using Corollary 3.3.

Theorem 5.5. *For deterministic languages the Borel hierarchy and the weak index hierarchy coincide (Fig. 4) and are decidable within the time of solving emptiness problem. For a deterministic automaton with n states, an equivalent minimal index automaton with $\mathcal{O}(n^2)$ states can be constructed effectively within the time of solving the emptiness problem.*

Proof. We will abuse the notation and write (ι, κ) to denote the class of languages recognized by weak (ι, κ)-automata. All the classes considered here are relativised to the deterministic languages.

By the two versions of the Gap Theorem we have the equality and decidability of the classes of the classes Π^0_3 and $(0,3)$.

Let us continue with the third level. Let us see that $\Sigma^0_3 = (1,4)$. We will show that both these classes are equal to the class of languages recognized by deterministic automata without a $(0,1)$-flower replicated by an accessible loop. If a deterministic automaton A does not contain this pattern, then it is equivalent to a weak $(1,4)$-automaton and by Corollary 3.3 recognizes a Σ^0_3 language. If A does contain this pattern, then by Proposition 4.2 it is

not Σ^0_3 and so is not equivalent to a weak $(1,4)$-automaton. The decidability follows easily, since checking for the pattern above can be done effectively (in polynomial time).

For the equality $\Pi^0_2 = (0,2)$, prove that both classes are equal to the class of languages recognized by deterministic automata without a $(0,1)$-flower. Proceed just like before, only use Proposition 5.1 instead of Proposition 5.4. Analogously, using Proposition 5.3, show that both Σ^0_2 and $(1,3)$ are equal to the class of languages recognized by deterministic automata admitting neither a $(1,2)$-flower nor a weak $(1,0)$-flower replicated by an accepting loop.

For the first level use the characterisation given by Proposition 4.4. The level zero is trivial. ∎

Acknowledgments

The author thanks Damian Niwiński for reading carefully a preliminary version of this paper and the anonymous referees for their helpful comments.

References

[1] A. Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. RAIRO-Theoretical Informatics and Applications **33** (1999) 329–339.

[2] A. Arnold, D. Niwiński. Fixed point characterisation of weak monadic logic definable sets of trees. *Tree Automata and Languages*, Elsevier 1992, 159–188.

[3] A. Arnold, D. Niwiński. Continuous separation of game languages. Manuscript, submitted, 2006.

[4] J. C. Bradfield. The modal mu-calculus alternation hierarchy is strict. Theoret. Comput. Sci. **195** (1998) 133–153.

[5] J. Duparc, F. Murlak. On the topological complexity of weakly recognizable tree languages. Proc. FCT 2007, LNCS 4639 (2007) 261-273.

[6] E. A. Emerson, C. S. Jutla. The complexity of tree automata and logics of programs. *Proc. FoCS '88*, IEEE Computer Society Press 1988, 328–337.

[7] O. Kupferman, M. Vardi. The weakness of self-complementation. *Proc. STACS '99*, LNCS 1563 (1999) 455–466.

[8] G. Lenzi. A hierarchy theorem for the mu-calculus. *Proc. ICALP '96*, LNCS 1099 (1996) 87–109.

[9] A. W. Mostowski. Hierarchies of weak automata and weak monadic formulas. Theoret. Comput. Sci. **83** (1991) 323-335.

[10] D. E. Muller, A. Saoudi, P. E. Schupp. Alternating automata. The weak monadic theory of the tree, and its complexity. *Proc. ICALP '86*, LNCS 226 (1986) 275–283.

[11] F. Murlak. On deciding topological classes of deterministic tree languages. *Proc. CSL '05*, LNCS 3634 (2005) 428–441.

[12] J. Neumann, A. Szepietowski, I. Walukiewicz. Complexity of weak acceptance conditions in tree automata. IPL 84 (2002) 181–187.

[13] D. Niwiński. On fixed point clones. *Proc. ICALP '86*, LNCS 226 (1986) 464–473.

[14] D. Niwiński, I. Walukiewicz. Relating hierarchies of word and tree automata. *Proc. STACS '98*, LNCS 1373 (1998) 320–331.

[15] D. Niwiński, I. Walukiewicz. A gap property of deterministic tree languages. Theoret. Comput. Sci. **303** (2003) 215–231.

[16] D. Niwiński, I. Walukiewicz. Deciding nondeterministic hierarchy of deterministic tree automata. *Proc. WoLLiC '04*, Electronic Notes in Theoret. Comp. Sci. 2005, 195–208.

[17] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. Trans. Amer. Soc. **141** (1969) 1–35.

[18] M. O. Rabin. Weakly definable relations and special automata. *Mathematical Logic and Foundations of Set Theory*, North-Holland 1970, 1–70.

[19] J. Skurczyński. The Borel hierarchy is infinite in the class of regular sets of trees. Theoret. Comput. Sci. **112** (1993) 413–418.

[20] W. Thomas. A hierarchy of sets of infinite trees. *Proc. Theoretical Computer Science*, LNCS 145 (1982) 335–342.
[21] K. Wagner. On ω-regular sets. Inform. and Control **43** (1979) 123–177.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 585-596
www.stacs-conf.org

A MAHLER'S THEOREM FOR FUNCTIONS FROM WORDS TO INTEGERS

JEAN-ÉRIC PIN [1] AND PEDRO V. SILVA [2]

[1] LIAFA, Université Paris-Diderot and CNRS, Case 7014, 75205 Paris Cedex 13, France.
E-mail address: Jean-Eric.Pin@liafa.jussieu.fr

[2] Centro de Matemática, Faculdade de Ciências, Universidade do Porto, R. Campo Alegre 687, 4169-007 Porto, Portugal.
E-mail address: pvsilva@fc.up.pt

Abstract. In this paper, we prove an extension of Mahler's theorem, a celebrated result of p-adic analysis. Mahler's original result states that a function from $\mathbb{N}$ to $\mathbb{Z}$ is uniformly continuous for the p-adic metric d_p if and only if it can be uniformly approximated by polynomial functions. We prove the same result for functions from A^* to $\mathbb{Z}$, where d_p is now the profinite metric defined by p-groups (pro-p metric).

This paper was originally motivated by two main research lines of automata theory, but resulted into an approximation theorem that goes far beyond our original project. We first present our original motivations and then describe our main result. Recall that a *variety of languages* is a class of regular languages closed under Boolean operations, left and right quotients and inverse morphisms.

1. Motivations

Our first motivation was the study of *regularity-preserving* functions f from A^* to B^*, in the following sense: if X is a regular language of B^*, then $f^{-1}(X)$ is a regular language of A^*. More generally, we were interested in functions *preserving a given variety of languages* $\mathcal{V}$: if X is a language of $\mathcal{V}$, then $f^{-1}(X)$ is also a language of $\mathcal{V}$. There is an important literature on the regular case [20, 6, 18, 12, 13, 2], including the authors recent paper [14]. A similar problem was also recently considered for formal power series [3]. A remarkable contribution to the second problem can be found in [16], where a characterization of sequential functions preserving aperiodic languages (respectively group-languages) is given.

1998 ACM Subject Classification: F.4.3, F.1.1.

Key words and phrases: p-adic topology, binomial coefficients, Mahler's theorem, p-group languages.

The authors acknowledge support from the AutoMathA programme of the European Science Foundation. The second author acknowledges support from Project ASA (PTDC/MAT/65481/2006) and C.M.U.P., financed by F.C.T. (Portugal) through the programmes POCTI and POSI, with national and European Community structural funds.

Our second motivation was the study of certain reductions. A fundamental idea of descriptive set theory is to use *continuous reductions* to classify topological spaces: given two sets X and Y, Y reduces to X if there exists a continuous function f such that $X = f^{-1}(Y)$. For Polish spaces, this gives rise to the Wadge hierarchy [21], nowadays entirely described and very well understood. Wagner [22] was the first to consider the restriction of the Wadge hierarchy to ω-rational languages. He proved in particular that the hierarchy doesn't change if continuous functions are replaced by sequential functions, a much more restricted class of functions (see [10, Chapter 5] for more details).

Our idea was to consider similar reductions for regular languages (of finite words this time). The first obstacle was to find an appropriate topology, but there is a natural candidate: the profinite topologies, a notion first introduced in [5]. Indeed, by Eilenberg's theorem, to each variety of languages $\mathcal{V}$ corresponds a unique variety of finite monoids $\mathbf{V}$, which in turn defines the pro-$\mathbf{V}$ topology [15]. We shall not give here the precise definition, but it suffices to know that, in the most interesting cases, this topology can be defined by a metric $d_{\mathbf{V}}$. Let us call $\mathbf{V}$-*reduction* a uniformly continuous function between the metric spaces $(A^*, d_{\mathbf{V}})$ and $(B^*, d_{\mathbf{V}})$. These $\mathbf{V}$-reductions define a hierarchy similar to the Wadge hierarchy among regular languages, which we would like to explore. Note that a different notion of reduction for regular languages was recently considered in [19]. The first author had very instructive discussions with Selivanov and Kunc in June 2006 on the comparison between these two reductions and this paper is partly motivated by this conversation.

Regularity-preserving functions and $\mathbf{V}$-reductions are actually strongly related. Indeed, one can show that a function from $(A^*, d_{\mathbf{V}})$ to $(B^*, d_{\mathbf{V}})$ is uniformly continuous if and only if, for every language L in $\mathcal{V}(B^*)$, $f^{-1}(L)$ belongs to $\mathcal{V}(A^*)$. This encouraging fact lead us to search for a more precise description of $\mathbf{V}$-reductions. However, apart from general results, not so much is known on pro-$\mathbf{V}$ topologies, except when $\mathbf{V}$ is a variety of finite groups. Among groups, the variety $\mathbf{G}_p$ of p-groups, where p is a given prime, is of special interest for two reasons. First, Eilenberg and Schützenberger gave a very nice description of the languages recognized by a p-group (see Proposition 2.2 below). Second, a special case of the metric d_p has been widely studied in mathematics: indeed, the free monoid over a one-letter alphabet is isomorphic to $\mathbb{N}$, and the metric d_p is known as the *p-adic metric*. The completion of the metric space $(\mathbb{N}, d_p)$ is the space of *p-adic numbers* and *p-adic analysis* is the branch of number theory that deals with functions of p-adic numbers [1, 17, 9].

Our main result takes advantage of this powerful mathematical framework to provide a characterization of the $\mathbf{G}_p$-reductions from A^* to $\mathbb{N}$, that is, the uniformly continuous functions from (A^*, d_p) to $(\mathbb{N}, d_p)$. It turns out that this characterization extends a celebrated result of number theory, Mahler's theorem (see `http://en.wikipedia.org/wiki/Mahler's_theorem`), giving our result a mathematical interest on its own. Our result states that a function from A^* to $\mathbb{N}$ is uniformly continuous for d_p if and only if it can be uniformly approximated by a sequence of polynomial functions. Before stating this result in a precise form, we need a few formal definitions.

2. The p-adic and pro-p topologies

In the sequel, A denotes a finite alphabet, A^* is the free monoid on A and 1 denotes the empty word.

Let p be a prime number. Recall that a *p-group* is a finite group whose order is a power of p. Let u and v be two words of A^*. A p-group G *separates* u and v if there is a monoid

morphism from A^* onto G such that $\varphi(u) \neq \varphi(v)$. One can show that any pair of distinct words can be separated by a p-group. We now set

$$r_p(u,v) = \min \{\ n \mid \text{there is a } p\text{-group of order } p^n \text{ separating } u \text{ and } v\ \}$$
$$d_p(u,v) = 2^{-r_p(u,v)}$$

with the usual convention $\min \emptyset = -\infty$ and $2^{-\infty} = 0$. One can show that d_p is an *ultrametric*, that is, satisfies the following properties, for all $u, v, w \in A^*$:

(1) $d_p(u,v) = 0$ if and only if $u = v$,
(2) $d_p(u,v) = d_p(v,u)$,
(3) $d_p(u,v) \leqslant \max(d_p(u,w), d_p(w,v))$

One can also show that the concatenation product on A^* is uniformly continuous for this metric. It follows that the completion of the metric space (A^*, d_p) is naturally equipped with a structure of monoid, which is in fact a compact group, called the *free pro-p group*. The topology defined by the metric d_p is usually called the *pro-p topology* in the literature.

There is a nice connection [11] between this topology and a generalization of the *binomial coefficients*. Let u and v be two words of A^*. Let $u = a_1 \cdots a_n$, with $a_1, \ldots, a_n \in A$. Then u is a *subword* of v if there exist $v_0, \ldots, v_n \in A^*$ such that $v = v_0 a_1 v_1 \ldots a_n v_n$. Following [4, 7], we define the binomial coefficient of u and v by setting

$$\binom{v}{u} = |\{(v_0, \ldots, v_n) \mid v = v_0 a_1 v_1 \ldots a_n v_n\}|\,.$$

Observe that if a is a letter, then $\binom{v}{a}$ is simply the number of occurrences of a in v, also denoted by $|v|_a$. Also note that if $A = \{a\}$, $u = a^n$ and $v = a^m$, then

$$\binom{v}{u} = \binom{m}{n}$$

and hence these numbers constitute a generalization of the classical binomial coefficients. The next proposition, whose proof can be found in [7, Chapter 6], summarizes the basic properties of the generalized binomial coefficients and can serve as an alternative definition.

Lemma 2.1. *Let $u, v \in A^*$ and $a, b \in A$. Then*

(1) $\binom{u}{1} = 1$,
(2) $\binom{u}{v} = 0$ *if* $|u| \leqslant |v|$ *and* $u \neq v$,
(3) $\binom{ua}{vb} = \begin{cases} \binom{u}{vb} & \text{if } a \neq b \\ \binom{u}{vb} + \binom{u}{v} & \text{if } a = b \end{cases}$

A third way to define the binomial coefficients is to use the *Magnus automorphism* of the algebra $\mathbb{Z}\langle A\rangle$ of polynomials in noncommutative indeterminates in A defined by $\mu(a) = 1 + a$ for all $a \in A$. One can show that, for all $u \in A^*$,

$$\mu(u) = \sum_{x \in A^*} \binom{u}{x} x \tag{2.1}$$

which leads to the formula

$$\binom{u_1 u_2}{x} = \sum_{x_1 x_2 = x} \binom{u_1}{x_1} \binom{u_2}{x_2} \tag{2.2}$$

The connection between the pro-p topology and the binomial coefficients comes from the characterization of the languages recognized by a p-group given by Eilenberg and Schützenberger (see [4, Theorem 10.1, p. 239]). Let us call a p-group language a language recognized by a p-group. Note that such a language is necessarily regular.

Proposition 2.2. *A language of A^* is a p-group language if and only if it is a Boolean combination of the languages*

$$L(x, r, p) = \{u \in A^* \mid \binom{u}{x} \equiv r \bmod p\},$$

for $0 \leqslant r < p$ and $x \in A^*$.

Let us set now

$$r'_p(u, v) = \min \left\{ |x| \;\middle|\; x \in A^* \text{ and } \binom{u}{x} \not\equiv \binom{v}{x} \pmod p \right\}$$
$$d'_p(u, v) = p^{-r'_p(u,v)}.$$

It is proved in [11, Theorem 4.4] that d'_p is an ultrametric uniformly equivalent to d_p. We shall use this result under a slightly different form.

Theorem 2.3. *A function $f : A^* \to B^*$ is uniformly continuous for d_p if and only if, for every regular language L of A^* recognized by a p-group, the language $f^{-1}(L)$ is also recognized by a p-group.*

Proof. Let L be a language recognized by a p-group G of order p^k. Then there exists a monoid morphism $\varphi : A^* \to G$ such that $L = \varphi^{-1}(\varphi(L))$. If f is uniformly continuous for d_p, there exists $n > 0$ such that if $r_p(u, v) \geqslant n$, then $r_p(f(u), f(v)) \geqslant k$. It follows in particular that $f(u)$ and $f(v)$ cannot be separated by G and hence $\varphi(f(u)) = \varphi(f(v))$.

Let $(\psi_i)_{i\in I}$ be the family of all monoid morphisms from A^* onto a p-group H_i of order $\leqslant p^n$. Let $\psi : A^* \to \prod_{i\in I} H_i$ be the morphism defined by $\psi(x) = (\psi_i(x))_{i\in I}$ and let H be the range of ψ. Then H is a p-group and if $\psi(u) = \psi(v)$, then $r_p(u, v) \geqslant n$ and thus $\varphi(f(u)) = \varphi(f(v))$. We claim that

$$\psi^{-1}(\psi(f^{-1}(L)) = f^{-1}(L)$$

First, $f^{-1}(L)$ is clearly a subset of $\psi^{-1}(\psi(f^{-1}(L))$. To prove the opposite inclusion, let $u \in \psi^{-1}(\psi(f^{-1}(L))$. Then $\psi(u) \in \psi(f^{-1}(L))$, that is, $\psi(u) = \psi(v)$ for some $v \in f^{-1}(L)$. It follows that $\varphi(f(u)) = \varphi(f(v))$ and since $f(v) \in L$, $f(u) \in \varphi^{-1}(\varphi(L))$ and finally $f(u) \in L$ since $L = \varphi^{-1}(\varphi(L))$. This proves the claim and shows that $f^{-1}(L)$ is a p-group language.

Suppose now that if L is a p-group language, then $f^{-1}(L)$ is also a p-group language. Let φ be a morphism from A^* onto a p-group G. For each $g \in G$, $\varphi^{-1}(g)$ is a p-group language and hence $f^{-1}(\varphi^{-1}(g))$ is recognized by a morphism $\psi_g : A^* \to H_g$ onto a p-group. Let $\psi : A^* \to \prod_{g\in G} H_g$ be the mapping defined by $\psi(x) = (\psi_g(x))_{g\in G}$ and let $H = \psi(A^*)$. Then H is also a p-group and if $\psi(u) = \psi(v)$, then $\psi_g(u) = \psi_g(v)$ for all $g \in G$. Since ψ_g recognizes $f^{-1}(\varphi^{-1}(g))$, it follows that $u \in f^{-1}(\varphi^{-1}(g))$ if and only if $v \in f^{-1}(\varphi^{-1}(g))$ and hence $\varphi(f(u)) = \varphi(f(v))$.

Now let $k \in \mathbb{N}$. If we consider all the morphisms φ from A^* onto a p-group of order $\leqslant p^k$, and take $n \in \mathbb{N}$ large enough so that every group H corresponding to φ has order $\leqslant p^n$, it follows that

$$r_p(u, v) > n \Rightarrow r_p(f(u), f(v)) > k$$

holds for all $u, v \in A^*$. This shows that f is uniformly continuous for d_p. ∎

In the case of a one-letter alphabet, A^* is isomorphic to the additive monoid $\mathbb{N}$ and the definition of d_p can be further simplified. If n is a non-zero integer, recall that the *p-adic valuation* of n is the integer

$$\nu_p(n) = \max\left\{k \in \mathbb{N} \mid p^k \text{ divides } n\right\}$$

By convention, $\nu_p(0) = +\infty$. The *p-adic norm* of n is the real number

$$|n|_p = p^{-\nu_p(n)}.$$

The p-adic norm satisfies the following axioms, for all $n, m \in \mathbb{N}$:

(1) $|n|_p \geqslant 0$,
(2) $|n|_p = 0$ if and only if $n = 0$,
(3) $|mn|_p = |m|_p|n|_p$,
(4) $|m + n|_p \leqslant \max\{|m|_p, |n|_p\}$.

Finally, the metric d_p can be defined by

$$d_p(u, v) = |u - v|_p.$$

and is known as the p-adic metric. Since $\mathbb{Z}$ is naturally embedded in the p-adic completion of $\mathbb{N}$, the definitions above can be readily extended to $\mathbb{Z}$. In the sequel, it will be more convenient to use the metric space $(\mathbb{Z}, d_p)$ in place of $(\mathbb{N}, d_p)$.

3. Mahler's expansions

The classical Stone-Weierstrass approximation theorem states that a continuous function defined on a closed interval can be uniformly approximated by a polynomial function. In particular, if a function f is infinitely differentiable in the neighbourhood of 0, it can be approximated, under some convergence conditions, by its Taylor polynomials

$$\sum_{n=0}^{k} \frac{f^{(n)}(0)}{n!}\, x^n$$

The p-adic analogue of these results is Mahler's theorem [8]. For a fixed $k \in \mathbb{N}$, the binomial polynomial function

$$u \to \binom{u}{k}$$

defines a uniformly continuous function from $(\mathbb{N}, d_p)$ to $(\mathbb{Z}, d_p)$. The *Mahler's expansion* of a function f from $\mathbb{N}$ to $\mathbb{Z}$ is defined as the series

$$\sum_{k=0}^{\infty} (\Delta^k f)(0) \binom{u}{k}$$

where Δ is the *difference operator*, defined by

$$(\Delta f)(u) = f(u + 1) - f(u)$$

Mahler's theorem states that f is uniformly continuous for d_p if and only if its Mahler's expansion converges uniformly to f. Of course, the most remarkable part of the theorem is the fact that any uniformly continuous function can be approximated by polynomial functions, in contrast to Stone-Weierstrass approximation theorem, which requires much stronger conditions.

For instance, if f is the Fibonacci sequence defined by $f(0) = f(1) = 1$ and $f(n) = f(n-1) + f(n-2)$ for $n \geqslant 2$, then

$$f(n) = \sum_{k=0}^{\infty} (-1)^{k+1} f(k) \binom{n}{k}$$

This function is not uniformly continuous for d_p for any choice of p. If $f(n) = r^n$, then

$$f(n) = \sum_{k=0}^{\infty} (r-1)^k \binom{n}{k}$$

and f is uniformly continuous for d_p if and only if p divides $r-1$.

The first step to extend Mahler's theorem to functions from words to integers is to define a suitable notion of Mahler's expansion for these functions.

Let $f : A^* \to \mathbb{Z}$ be a function. For each letter a, we define the difference operator Δ^a by

$$(\Delta^a f)(u) = f(ua) - f(u)$$

One can now define inductively an operator Δ^w for each word $w \in A^*$ by setting $(\Delta^1 f)(u) = f(u)$, and for each letter $a \in A$,

$$(\Delta^{aw} f)(u) = (\Delta^a(\Delta^w f))(u).$$

It is easy to see that these operators can also be defined directly by setting

$$\Delta^w f(u) = \sum_{0 \leqslant |x| \leqslant |w|} (-1)^{|w|+|x|} \binom{w}{x} f(ux) \tag{3.1}$$

For instance, $\Delta^{aab} f(u) = -f(u) + 2f(ua) + f(ub) - f(uaa) - 2f(uab) + f(uaab)$.
For a fixed $v \in A^*$, the function

$$u \to \binom{u}{v}$$

from A^* to $\mathbb{Z}$ which maps a word u to the binomial coefficient $\binom{u}{v}$ is uniformly continuous for d_p. This family of functions, for v ranging over A^*, is *locally finite* in the sense that, for each $u \in A^*$, the binomial coefficient $\binom{u}{v}$ is null for all but finitely many words v. In particular, if $(m_v)_{v \in A^*}$ is a family of integers, there is a well-defined function from A^* to $\mathbb{Z}$ defined by the formula

$$f(u) = \sum_{v \in A^*} m_v \binom{u}{v}$$

We can now state our first result, which doesn't require any assumption on f.

Theorem 3.1. *Let $f : A^* \to \mathbb{Z}$ be an arbitrary function. Then there exists a unique family $\langle f, v \rangle_{v \in A^*}$ of integers such that, for all $u \in A^*$,*

$$f(u) = \sum_{v \in A^*} \langle f, v \rangle \binom{u}{v} \tag{3.2}$$

This family is given by

$$\langle f, v \rangle = (\Delta^v f)(1) = \sum_{0 \leqslant |x| \leqslant |v|} (-1)^{|v|+|x|} \binom{v}{x} f(x) \tag{3.3}$$

Proof. First observe that, according to (3.1)

$$(\Delta^v f)(1) = \sum_{0 \leqslant |x| \leqslant |v|} (-1)^{|v|+|x|} \binom{v}{x} f(x) \tag{3.4}$$

Thus

$$\begin{aligned}\sum_{v \in A^*} (\Delta^v f)(1) \binom{u}{v} &= \sum_{v \in A^*} \sum_{|x| \leqslant |v|} (-1)^{|v|+|x|} \binom{v}{x} \binom{u}{v} f(x) \\ &= \sum_{x \in A^*} (-1)^{|u|+|x|} \left(\sum_{0 \leqslant |v| \leqslant |u|} (-1)^{|v|+|u|} \binom{u}{v} \binom{v}{x} \right) f(x) \\ &= f(u)\end{aligned}$$

in view of the following relation from [7, Corollary 6.3.8]:

$$\sum_{0 \leqslant |v| \leqslant |u|} (-1)^{|u|+|v|} \binom{u}{v} \binom{v}{w} = \begin{cases} 1 & \text{if } u = w \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Uniqueness of the coefficients $\langle f, v \rangle$ follows inductively from the formula

$$\langle f, u \rangle = f(u) - \sum_{0 \leqslant |v| < |u|} \langle f, v \rangle \binom{u}{v},$$

a straightforward consequence of (3.2). ∎

The series defined by (3.2) is called the *Mahler's expansion* of f.

For instance, let $f : \{0,1\}^* \to \mathbb{N}$ be the function mapping a binary word onto its value as a binary number. Thus $f(010111) = f(10111) = 23$. Then one has

$$(\Delta^v f) = \begin{cases} f + 1 & \text{if } |v|_1 > 0 \\ f & \text{otherwise} \end{cases}$$

$$(\Delta^v f)(\varepsilon) = \begin{cases} 1 & \text{if } |v|_1 > 0 \\ 0 & \text{otherwise} \end{cases}$$

Thus, if $u = 01001$, one gets

$$f(u) = \binom{u}{1} + \binom{u}{10} + \binom{u}{11} + \binom{u}{100} + \binom{u}{101} + \binom{u}{1001} = 2 + 2 + 1 + 1 + 2 + 1 = 9$$

4. Mahler polynomials

A function $f : A^* \to \mathbb{Z}$ is a *Mahler polynomial* if its Mahler's expansion has *finite support*, that is, if the number of nonzero coefficients $\langle f, v \rangle$ is finite. In this section, we prove in particular that Mahler polynomials are closed under addition and product. We first introduce a convenient combinatorial operation, the infiltration product. We follow the presentation of [7].

Let $\mathbb{Z}\langle\langle A \rangle\rangle$ be the ring of formal power series in noncommutative indeterminates in A. Any series s is written as a formal sum $s = \sum_{u \in A^*} \langle s, u \rangle u$, a notation not to be confused with

our notation for the Mahler's expansion. The *infiltration product* is the binary operation on $\mathbb{Z}\langle\langle A\rangle\rangle$, denoted by $\uparrow$ and defined inductively as follows:

for all $u \in A^*$,

$$u \uparrow 1 = 1 \uparrow u = u, \tag{4.1}$$

for all $u, v \in A^*$, for all $a, b \in A$

$$ua \uparrow bv = \begin{cases} (u \uparrow vb)a + (ua \uparrow v)b + (u \uparrow v)a & \text{if } a = b \\ (u \uparrow vb)a + (ua \uparrow v)b & \text{otherwise} \end{cases} \tag{4.2}$$

for all $s, t \in \mathbb{Z}\langle\langle A\rangle\rangle$,

$$s \uparrow t = \sum_{u,v \in A^*} \langle s, u\rangle\langle t, v\rangle(u \uparrow v) \tag{4.3}$$

Intuitively, the coefficient $\langle u \uparrow v, x\rangle$ is the number of pairs of subsequences of x which are respectively equal to u and v and whose union gives the whole sequence x. For instance,

$$ab \uparrow ab = ab + 2aab + 2abb + 4aabb + 2abab$$
$$ab \uparrow ba = aba + bab + abab + 2abba + 2baab + baba$$

Also note that $\langle u \uparrow v, u\rangle = \binom{u}{v}$. We shall need the following relation (see [7, p.131]). For all $v_1, v_2 \in A^*$,

$$\binom{u}{v_1}\binom{u}{v_2} = \sum_{x \in A^*} \langle v_1 \uparrow v_2, x\rangle \binom{u}{x} \tag{4.4}$$

Formula 4.4 leads to an explicit computation of the Mahler's expansion of the product of two functions.

Proposition 4.1. *Let f and g be two functions from A^* to $\mathbb{N}$. Then the coefficients of the Mahler's expansion of fg are given by the formula:*

$$\langle fg, x\rangle = \sum_{v_1, v_2 \in A^*} \langle f, v_1\rangle\langle g, v_2\rangle\langle v_1 \uparrow v_2, x\rangle$$

Proof. Indeed, if $f(u) = \sum_{v \in A^*} \langle f, v\rangle \binom{u}{v}$ and $g(u) = \sum_{v \in A^*} \langle g, v\rangle \binom{u}{v}$, then

$$fg(u) = \sum_{v_1, v_2 \in A^*} \langle f, v_1\rangle\langle g, v_2\rangle \binom{u}{v_1}\binom{u}{v_2}$$

and the result follows by Formula (4.4). ∎

It is now easy to prove the result announced at the beginning of this section.

Proposition 4.2. *Mahler polynomials form a subring of the ring of all functions from A^* to $\mathbb{Z}$ for addition and multiplication.*

Proof. It is clear that the difference of two Mahler polynomials is a Mahler polynomial. Further Proposition 4.1 shows that Mahler polynomials are closed under product. ∎

5. Mahler's theorem

We are now ready to state our main result, which extends Mahler's theorem. In this section, uniform continuity always refers to the metric d_p.

Theorem 5.1. *Let $f(u) = \sum_{v\in A^*}\langle f, v\rangle\binom{u}{v}$ be the Mahler's expansion of a function from A^* to $\mathbb{Z}$. Then f is uniformly continuous if and only if $\lim_{|v|\to\infty}|\langle f, v\rangle|_p = 0$.*

Proof. Suppose that $\lim_{|v|\to\infty}|\langle f, v\rangle|_p = 0$. Then there exists $s \in \mathbb{N}$ such that, if $|v| \geqslant s$, $\nu_p(\langle f, v\rangle) \geqslant r$. Setting

$$g(u) = \sum_{|v|<s}\langle f, v\rangle\binom{u}{v} \text{ and } h(u) = \sum_{|v|\geqslant s}\langle f, v\rangle\binom{u}{v}$$

we get $f = g + h$. Further p^r divides $\langle f, v\rangle$ for $|v| \geqslant s$. Since g is a Mahler polynomial, it is uniformly continuous and there exists $t \in \mathbb{N}$ such that $d_p(u, u') \leqslant p^{-t}$ implies $g(u) \equiv g(u') \pmod{p^r}$ and hence $f(u) \equiv f(u') \pmod{p^r}$. Thus f is uniformly continuous.

This proves the easy direction of the theorem. The key argument for the opposite direction is the following approximation result.

Theorem 5.2. *Let $f : A^* \to \mathbb{N}$ be a uniformly continuous function. Then there exists a Mahler polynomial P such that, for all $u \in A^*$, $f(u) \equiv P(u) \pmod{p}$.*

Proof. We first prove the theorem for some characteristic functions related to the binomial coefficients. The precise role of these functions will appear in the course of the main proof.

Let $x \in A^*$ and let s be an integer such that $0 \leqslant s < p$. Let $\chi_{s,x} : A^* \to \mathbb{N}$ be the function defined by

$$\chi_{s,x}(u) = \begin{cases} 1 & \text{if } \binom{u}{x} \equiv s \pmod{p} \\ 0 & \text{otherwise} \end{cases}$$

Lemma 5.3. *There is a Mahler polynomial $P_{s,x}$ such that, for all $u \in A^*$, $\chi_{s,x}(u) \equiv P_{s,x}(u) \pmod{p}$.*

Proof. Let

$$P_{s,x}(u) = -\frac{\left[\binom{u}{x}\right]\left[\binom{u}{x}-1\right]\cdots\left[\binom{u}{x}-(p-1)\right]}{\binom{u}{x}-s}$$

Then $P_{s,x}$ is a Mahler polynomial by Proposition 4.2. If $\binom{u}{x} \not\equiv s \pmod{p}$, then $P_{s,x}(u) \equiv 0 \pmod{p}$. If $\binom{u}{x} \equiv s \pmod{p}$, then by Al-Haytham's theorem,

$$P_{s,x}(u) \equiv -(p-1)! \equiv 1 \pmod{p}$$

It follows that $P_{s,x}(u) \equiv \chi_{s,x}(u) \pmod{p}$ in all cases. ∎

We now prove Theorem 5.2. Since f is uniformly continuous, there exists a positive integer n such that if, for $0 \leqslant |x| \leqslant n$,

$$\binom{u}{x} \equiv \binom{v}{x} \pmod{p}$$

then

$$f(u) \equiv f(v) \pmod{p}$$

It follows that the value of $f(u)$ modulo p depends only on the residues modulo p of the family $\left\{\binom{u}{x}\right\}_{0\leqslant|x|\leqslant n}$.

Let C be the set of all families $r = \{r_x\}_{0 \leqslant |x| \leqslant n}$ such that $0 \leqslant r_x < p$. For $0 \leqslant i < p$, let C_i be the set of all families r of C satisfying the following condition:

$$\text{if, for } 0 \leqslant |x| \leqslant n, \binom{u}{x} \equiv r_x \pmod{p}, \text{ then } f(u) \equiv i \pmod{p} \tag{5.1}$$

The sets $(C_i)_{0 \leqslant i < p}$ are pairwise disjoint and their union is C. We claim that, for all $u \in A^*$,

$$f(u) \equiv \sum_{0 \leqslant i < p} iP_i(u) \pmod{p} \tag{5.2}$$

where P_i is the Mahler polynomial

$$P_i = \sum_{r \in C_i} \prod_{0 \leqslant |x| \leqslant n} P_{r_x, x} \tag{5.3}$$

First consider, for $r \in C$, the characteristic function

$$\chi_r(u) = \prod_{0 \leqslant |x| \leqslant n} \chi_{r_x, x}(u)$$

By construction, χ_r is defined by

$$\chi_r(u) = \begin{cases} 1 & \text{if, for } 0 \leqslant |x| \leqslant n, \binom{u}{x} \equiv r_x \pmod{p} \\ 0 & \text{otherwise} \end{cases}$$

and it follows from (5.1) and from the definition of C_i that

$$f(u) \equiv \sum_{0 \leqslant i < p} \Big(i \sum_{r \in C_i} \chi_r(u) \Big) \pmod{p} \tag{5.4}$$

Now Lemma 5.3 gives immediately

$$\chi_r(u) \equiv \prod_{0 \leqslant |x| \leqslant n} P_{r_x, x}(u) \pmod{p} \tag{5.5}$$

and thus (5.2) follows now from (5.3), (5.4) and (5.5). The result follows, since

$$P = \sum_{0 \leqslant i < p} iP_i(u)$$

is a Mahler polynomial. ∎

Theorem 5.2 can be extended as follows.

Corollary 5.4. *Let $f : A^* \to \mathbb{N}$ be a uniformly continuous function. Then, for each positive integer r, there exists a Mahler polynomial P_r such that, for all $u \in A^*$, $f(u) \equiv P_r(u) \pmod{p^r}$.*

Proof. We prove the result by induction on r. For $r = 1$, the result follows from Theorem 5.2. If the result holds for r, there exists a Mahler polynomial P_r such that, for all $u \in A^*$, $f(u) - P_r(u) \equiv 0 \pmod{p^r}$. Let $g = f - P_r$. Since g is uniformly continuous, there exists a positive integer n such that if $\binom{u}{x} \equiv \binom{v}{x} \pmod{p}$ for $|x| \leqslant n$, then $g(u) \equiv g(v) \pmod{p^{2r}}$. It follows that $\frac{1}{p^r} g(u) \equiv \frac{1}{p^r} g(v) \pmod{p^r}$, and thus $\frac{1}{p^r} g$ is uniformly continuous.

Applying Theorem 5.2 to $\frac{1}{p^r} g$, we get a Mahler polynomial P such that, for all $u \in A^*$,

$$\frac{1}{p^r} g(u) \equiv P(u) \pmod{p}$$

Setting $P_{r+1} = P_r + p^r P$, we obtain finally

$$f(u) \equiv P_{r+1}(u) \pmod{p^{r+1}}$$

which concludes the proof. ∎

We now conclude the proof of Theorem 5.1. For each positive integer r, there exists a Mahler polynomial P_r such that, for all $u \in A^*$, $f(u) \equiv P_r(u) \pmod{p^r}$. Using (3.3) to compute explicitly the coefficients $\langle f - P_r, v\rangle$, we obtain

$$\langle f - P_r, v\rangle \equiv 0 \pmod{p^r}$$

Since P_r is a polynomial, there exists an integer n_r such that for all $v \in A^*$ such that $|v| \geqslant n$, $\langle P_r, v\rangle = 0$. It follows $|\langle f, v\rangle|_p < p^{-r}$ and thus $\lim_{|v|\to\infty} |\langle f, v\rangle|_p = 0$. ∎

Mahler's theorem is often presented as an interpolation result (see for instance [9, p. 57]). This can also be extended to functions from words to integers. Given a family of integers $(c_v)_{v\in A^*}$, one can ask whether there is a (uniformly) continuous function f from the free pro-p group to $\mathbb{Z}$ such that $f(v) = c_v$. Then answer is yes if and only if $\lim_{|v|\to\infty} |m_v|_p = 0$, where $m_v = \sum_{0\leqslant|x|\leqslant|v|} (-1)^{|v|+|x|} \binom{v}{x} c_x$.

6. Conclusion

We proved an extension of Mahler's theorem for functions from words to integers. It would be interesting to find a suitable extension for functions from words to words. It would also be interesting to see whether other results from p-adic analysis can be extended to the word case.

Acknowledgement

The authors would like to thank Daniel Barsky and Gilles Christol for pointing out several references.

References

[1] Y. Amice, Interpolation p-adique, *Bull. Soc. Math. France* **92** (1964), 117–180.
[2] J. Berstel, L. Boasson, O. Carton, B. Petazzoni and J.-E. Pin, Operations preserving recognizable languages, *Theoret. Comput. Sci.* **354** (2006), 405–420.
[3] M. Droste and G.-Q. Zhang, On transformations of formal power series, *Inform. and Comput.* **184**,2 (2003), 369–383.
[4] S. Eilenberg, *Automata, Languages and Machines*, vol. B, Academic Press, New York, 1976.
[5] M. Hall, Jr., A topology for free groups and related groups, *Ann. of Math. (2)* **52** (1950), 127–139.
[6] S. R. Kosaraju, Regularity preserving functions, *SIGACT News* **6 (2)** (1974), 16–17. Correction **6 (3)** (1974), 22.
[7] M. Lothaire, *Combinatorics on words*, *Cambridge Mathematical Library*, Cambridge University Press, Cambridge, 1997.
[8] K. Mahler, An interpolation series for continuous functions of a p-adic variable., *J. Reine Angew. Math.* **199** (1958), 23–34. Correction **208** (1961), 70–72.
[9] M. R. Murty, *Introduction to p-adic analytic number theory*, *Studies in Advanced Mathematics*, American Mathematical Society, Cambridge, 2002.
[10] D. Perrin and J.-E. Pin, *Infinite Words*, *Pure and Applied Mathematics* vol. 141, Elsevier, 2004. ISBN 0-12-532111-2.
[11] J.-E. Pin, Topologie p-adique sur les mots, *Journal de théorie des nombres de Bordeaux* **5** (1993), 263–281.

[12] J.-E. Pin and J. Sakarovitch, Operations and transductions that preserve rationality, in *6th GI Conference*, Berlin, 1983, pp. 617–628, *Lect. Notes Comp. Sci.* n° 145, Lect. Notes Comp. Sci.
[13] J.-E. Pin and J. Sakarovitch, Une application de la représentation matricielle des transductions, *Theoret. Comput. Sci.* **35** (1985), 271–293.
[14] J.-E. Pin and P. V. Silva, A topological approach to transductions, *Theoret. Comput. Sci.* **340** (2005), 443–456.
[15] J.-E. Pin and P. Weil, Uniformities on free semigroups, *International Journal of Algebra and Computation* **9** (1999), 431–453.
[16] C. Reutenauer and M.-P. Schützenberger, Variétés et fonctions rationnelles, *Theoret. Comput. Sci.* **145**,1-2 (1995), 229–240.
[17] A. M. Robert, *A course in p-adic analysis*, *Graduate Texts in Mathematics* vol. 198, Springer-Verlag, New York, 2000.
[18] J. I. Seiferas and R. McNaughton, Regularity-preserving relations, *Theoret. Comp. Sci.* **2** (1976), 147–154.
[19] V. L. Selivanov and K. W. Wagner, A reducibility for the dot-depth hierarchy, *Theoret. Comput. Sci.* **345**,2-3 (2005), 448–472.
[20] R. E. Stearns and J. Hartmanis, Regularity preserving modifications of regular expressions, *Information and Control* **6** (1963), 55–69.
[21] W. Wadge, *Reducibility and determinateness in the Baire space*, PhD thesis, University of California, Berkeley, 1983.
[22] K. Wagner, On ω-regular sets, *Inform. and Control* **43**,2 (1979), 123–177.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 597-608
www.stacs-conf.org

DISTINGUISHING SHORT QUANTUM COMPUTATIONS

BILL ROSGEN [1]

[1] Institute for Quantum Computing and School of Computer Science, University of Waterloo
E-mail address: wrosgen@iqc.ca

Abstract. Distinguishing logarithmic depth quantum circuits on mixed states is shown to be complete for QIP, the class of problems having quantum interactive proof systems. Circuits in this model can represent arbitrary quantum processes, and thus this result has implications for the verification of implementations of quantum algorithms. The distinguishability problem is also complete for QIP on constant depth circuits containing the unbounded fan-out gate. These results are shown by reducing a QIP-complete problem to a logarithmic depth version of itself using a parallelization technique.

1. Introduction

Much of the difficulty in implementing quantum algorithms in practice is that qubits quickly decohere upon interacting with the environment. This entanglement destroying process limits the length of the computations that can be realized by experiment. Implementing quantum algorithms as circuits of low depth can provide a way to perform as much computation as possible within the limited time available, and for this reason there is significant interest in finding short quantum circuits for important problems.

Log-depth quantum circuits have been found for several significant problems including the approximate quantum Fourier transform [3] and the encoding and decoding operations for many quantum error correcting codes [10]. In addition to these applications, a procedure for parallelizing to log-depth a large class of quantum circuits has recently been discovered [2]. These examples demonstrate the surprising power of short quantum circuits.

Much of the work on quantum circuits is done in the standard model of unitary quantum circuits on pure states. In this paper a slightly different model of computation is considered: the model of mixed state quantum circuits, introduced by Aharonov, Kitaev, and Nisan [1]. While much of the previous complexity-theoretic work on short quantum circuits has been in the unitary model [4, 6], there has also been work outside of this model [13]. There are several advantages to considering the more general model of mixed state circuits. The primary advantage is that the mixed state model is able to capture any process allowed by quantum mechanics, so that results on this model may have implications for experimental work in quantum computing. The problem of distinguishing circuits may thus be thought of as the problem of distinguishing potentially noisy physical processes. As an example,

Key words and phrases: quantum information, computational complexity, quantum circuits, quantum interactive proof systems.

finding an error in an implementation of a quantum algorithm is simply the problem of distinguishing the constructed circuit from one that is known to be correct.

Unfortunately, in this paper it is shown that the apparent power of short quantum computations comes with a price: logarithmic depth quantum circuits are *exactly* as difficult to distinguish as polynomial depth quantum circuits. This equivalence implies the surprising result that distinguishing log-depth quantum computations is complete for the class QIP, the set of all problems that have quantum interactive proof systems. As PSPACE $\subseteq$ QIP $\subseteq$ EXP [8], this result also implies that the problem is PSPACE-hard.

The result on circuit distinguishability is shown using the closely related problem of determining if two circuits can be made to output states that are close together. This problem was introduced by Kitaev and Watrous [8] who show it to be both complete for QIP and contained in EXP. The main result of the present paper is obtained by reducing an instance of this problem of polynomial depth to an equivalent instance of logarithmic depth. This demonstrates that the problem of close images remains complete for QIP even under a logarithmic depth restriction. The hardness of distinguishing short quantum circuits is then demonstrated by a modification to the argument in [12] to show that the equivalence of close images problem and the distinguishability holds even for log-depth circuits.

The remainder of this paper is organized as follows. In the next section, some of the notation and results that will be needed are summarized. This is followed by Section 3, where the complete problems for QIP are discussed. In Section 4 the reduction from the polynomial depth to logarithmic depth versions of the close images problem is given, and the correctness of this construction is shown in Section 5. The equivalence between the log-depth close images problem and the problem of distinguishing log-depth computations is discussed in Section 6.

2. Preliminaries

This section outlines some of the definitions and results that will be used throughout the paper. For a more thorough treatment of the concepts introduced here see [9] and [11].

Throughout the paper scripted letters such as $\mathcal{H}$ will refer to finite dimensional Hilbert spaces, $\mathbf{D}(\mathcal{H})$ will denote the set of all density matrices on $\mathcal{H}$, and $\mathbf{U}(\mathcal{H}, \mathcal{K})$ will denote the norm-preserving linear operators from $\mathcal{H}$ to $\mathcal{K}$. The proof of the main result will make extensive use of two notions of distance between quantum states. The first of these is the fidelity. The *fidelity* between two positive semidefinite operators X and Y on a space $\mathcal{H}$ can be defined as

$$F(X, Y) = \max\{|\langle\phi|\psi\rangle| : |\phi\rangle, |\psi\rangle \in \mathcal{H} \otimes \mathcal{K}, \operatorname{tr}_{\mathcal{K}} |\phi\rangle\langle\phi| = X, \operatorname{tr}_{\mathcal{K}} |\psi\rangle\langle\psi| = Y\}.$$

This definition is known as Uhlmann's Theorem, and it is used here as it is more directly applicable to the task at hand than the usual definition. As any purification of a state necessarily purifies the partial trace of that state, this equation implies that the fidelity is nondecreasing under the partial trace. This property is known as *monotonicity* and can be stated more formally as $F(X, Y) \leq F(\operatorname{tr}_{\mathcal{K}} X, \operatorname{tr}_{\mathcal{K}} Y)$ where X, Y are positive semidefinite operators on $\mathcal{H} \otimes \mathcal{K}$. The final property of the fidelity that will be needed is the result that the maximum fidelity of any outputs of two transformations is multiplicative with respect to the tensor product. This result can be found in [9] (see Problem 11.10 and apply the multiplicativity of the diamond norm with respect to the tensor product).

Theorem 2.1 (Kitaev, Shen, and Vyalyi [9]). *For any completely positive transformations* $\Phi_1, \Phi_2, \Psi_1, \Psi_2$ *on states in* $\mathcal{H}$

$$\max_{\rho,\xi\in\mathbf{D}(\mathcal{H}\otimes\mathcal{H})} \mathrm{F}\left((\Phi_1\otimes\Phi_2)(\rho),(\Psi_1\otimes\Psi_2)(\xi)\right) = \prod_{i=1,2}\max_{\rho,\xi\in\mathbf{D}(\mathcal{H})} \mathrm{F}(\Phi_i(\rho),\Psi_i(\xi))$$

The second notion of distance that will be used is the *trace norm*, which can be defined for any linear operator X by $\|X\|_{\mathrm{tr}} = \mathrm{tr}\sqrt{X^*X}$, or equivalently as the sum of the singular values of X. This quantity is a norm, and so in particular it satisfies the triangle inequality. Similar to the fidelity, the trace norm is monotone under the partial trace, though in this case the trace norm is non-increasing under this operation. The proofs that follow will make essential use of the Fuchs-van de Graaf Inequalities [5] that relate the trace norm and the fidelity. For any density operators ρ and ξ on the same space, these inequalities are

$$1-\mathrm{F}(\rho,\xi) \leq \frac{1}{2}\|\rho-\xi\|_{\mathrm{tr}} \leq \sqrt{1-\mathrm{F}(\rho,\xi)}.$$

In addition to these measures on quantum states, it will be helpful to have a distance measure on quantum transformations. One such measure is the *diamond norm*, which for a completely positive transformation Φ on density operators on $\mathcal{H}$ is given by

$$\|\Phi\|_{\diamond} = \sup_{\rho\in\mathbf{D}(\mathcal{H}\otimes\mathcal{H})} \left\|(\Phi\otimes I_{\mathbf{L}(\mathcal{H})})(\rho)\right\|_{\mathrm{tr}}.$$

This norm is essential when considering transformations as it represents the distinguishability of two transformations when a reference system is taken into account. The simple supremum of the trace norm over all inputs to the channel is not stable under the addition of a reference system, and so the diamond norm is used in place of the simpler one. More properties and a more thorough definition of this norm can be found in [9].

The circuit model that will be used in this paper is the mixed state model introduced by Aharonov, Kitaev, and Nisan [1]. Circuits in this model are composed of qubits that are acted upon by arbitrary trace preserving and completely positive operations. This model allows for non-unitary operations, such as measurement or the introduction of ancillary qubits, to occur in the middle of the circuit. It is important to note that this model captures any physical process that quantum mechanics allows, and so in particular, any computation that can be done on mixed states with measurements can be represented in this model. Fortunately this model is polynomially equivalent to the standard model of unitary quantum circuits (with ancilla) followed by measurement, as shown in [1]. This will allow us to consider only circuits composed of unitary gates from some finite basis of one and two qubit gates with the additional operations of introducing qubits in the $|0\rangle$ state and measuring in the computational basis. This restriction can be strengthened, again with no loss of generality, to assume that all ancillary qubits are introduced at the start of the circuit and that all measurements are performed at the end.

We will often add to this circuit model one additional gate: the unbounded fan-out gate. This gate, in constant depth, applies a controlled-not operation from one qubit to an arbitrary number of output qubits. It is not clear that this gate is a reasonable choice in a standard basis of gates, and so it will be clearly marked when this gate is allowed into the circuit model under consideration. As an example of the power of this gate it can be used to build a constant depth circuit for the approximate quantum Fourier transform [7]. This gate is considered here for the sole reason that if it is included in the standard set of gates, the main result will also hold for constant depth circuits.

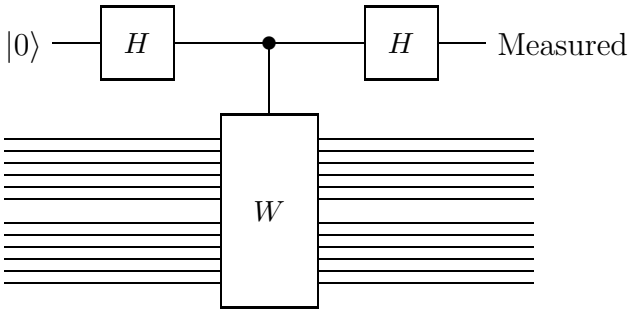


Figure 1: A circuit implementing the swap test.

For spaces $\mathcal{H}$ and $\mathcal{K}$ of the same dimension, we use $W \in \mathbf{U}(\mathcal{H} \otimes \mathcal{K}, \mathcal{H} \otimes \mathcal{K})$ to represent the operation that swaps the states in the two spaces. As W is a permutation matrix when expressed in the computational basis, and the permutation that it encodes is composed exclusively of transpositions, the swap operation is both hermitian and unitary. Furthermore, W can easily be implemented in constant depth, as all of the required transpositions can be performed in parallel. This operator is the essential component of the *swap test*, where a controlled W operation is used to determine how close two states are to each other. A circuit performing the swap test is given in Figure 1, where the measurement is performed in the computational basis. Another way to view the swap test is as a projective measurement onto the symmetric and antisymmetric subspaces. The projections in this measurement are given by $(I + W)/2$ and $(I - W)/2$. This formulation of the swap test is equivalent to the circuit presented in Figure 1.

It is not immediately clear how a controlled operation on n qubits, such as the controlled-swap operation used in the swap test, can be performed in depth logarithmic in n. The straightforward implementation requires using one control qubit to control each of the gates in the operation. However, Moore and Nilsson [10] give a simple construction that allows such an operation to be performed in log-depth.

Proposition 2.2 (Moore and Nilsson)**.** *Any log depth operation on n qubits controlled by one qubit can be implemented in $O(\log n)$ depth with $O(n)$ ancillary qubits.*

Moore and Nilsson prove this only for the constant depth case, but the method of proof that they use immediately extends to the log depth case. They prove this proposition by using a tree of $\log n$ controlled-not operations to 'duplicate' the control qubit. These copies can then be used to control the remaining operations, with each control qubit used at most a logarithmic number of times. This proposition, as an example, implies that the swap test circuit on n qubits shown in Figure 1 can be implemented in depth $O(\log n)$.

If the fan-out gate is allowed into the standard basis of gates, then controlled operations can be performed with only constant depth overhead. A circuit that performs this can be obtained by simply using one fan-out gate to make n copies (in the computational basis) of the control qubit onto ancillary qubits. These 'copies' may then be used to control each of the n operations, with a final application of the fan-out gate to restore the ancillary qubits to the $|0\rangle$ state. As controlled operations will be the only place that the circuits constructed here exceed constant depth, this will allow the proof of the main result for constant depth circuits with fan-out.

3. Complete Problems for QIP

The Close Images problem, defined and shown to be complete for QIP in [8] can be stated as follows.

Close Images. *For constants $0 < b < a \leq 1$, the input consists of quantum circuits Q_1 and Q_2 that implement transformations from $\mathcal{H}$ to $\mathcal{K}$. The promise problem is to distinguish the two cases:*

Yes: $\mathrm{F}(Q_1(\rho), Q_2(\xi)) \geq a$ *for some* $\rho, \xi \in \mathbf{D}(\mathcal{H})$,
No: $\mathrm{F}(Q_1(\rho), Q_2(\xi)) \leq b$ *for all* $\rho, \xi \in \mathbf{D}(\mathcal{H})$.

This is simply the problem of determining if there are inputs to Q_1 and Q_2 that cause them to output states that are nearly the same. It will be helpful to abbreviate the name of this problem as $\mathsf{CI}_{a,b}$.

A closely related problem is that of distinguishing two quantum circuits. This problem was introduced and shown complete for QIP in [12].

Quantum Circuit Distinguishability. *For constants $0 \leq b < a \leq 2$, the input consists of quantum circuits Q_1 and Q_2 that implement transformations from $\mathcal{H}$ to $\mathcal{K}$. The promise problem is to distinguish the two cases:*

Yes: $\|Q_1 - Q_2\|_\diamond \geq a$,
No: $\|Q_1 - Q_2\|_\diamond \leq b$.

Less formally, this problem asks: is there an input density matrix ρ on which the circuits Q_1 and Q_2 can be made to act differently? This problem will be referred to as $\mathsf{QCD}_{a,b}$.

It is our aim to prove that these problems remain complete for QIP when restricted to circuits Q_1 and Q_2 that are of depth logarithmic in the number of input qubits. This will be achieved in the case of perfect soundness error, i.e. $a = 1, 2$ in the above problem definitions. Both of these problem remain complete for QIP in this case. This restriction serves only to make these problems easier, as distinguishing the two cases for a weaker promise can only be more difficult, so the results of this paper will also imply the hardness of the more general problems. The log-depth versions of these problems will be referred to as Log-depth $\mathsf{CI}_{1,b}$ and Log-depth $\mathsf{QCD}_{2,b}$, and since these are restrictions of QIP-complete problems it is clear that they are also in QIP. Similarly, the abbreviations Const-depth $\mathsf{CI}_{1,b}$ and Const-depth $\mathsf{QCD}_{2,b}$ for the versions of these problems on constant-depth circuits will be convenient.

4. Log-Depth Construction

In this section the reduction from the general $\mathsf{CI}_{1,b}$ problem to the log-depth restriction of the problem is described. The general idea behind the construction is to simply slice the circuits of an instance of $\mathsf{CI}_{1,b}$ into logarithmic-depth pieces and run them in parallel. These circuits will require more input, but if each piece of the circuit is given as input the same state output by the previous piece, then the output of the last piece of the circuit will be equal to the output of the original circuit. This may not be the case if the intermediate inputs are not the outputs of the previous pieces, and so additional tests that ensure these inputs are at least close to the desired states are required.

To describe the reduction, let Q_1 and Q_2 be the circuits from an instance of $\mathsf{CI}_{1,b}$, and let n be the size (number of gates) of Q_1 and Q_2 (by padding the smaller circuit, if necessary). In order to perform the slicing of the circuit into pieces it is assumed that Q_1

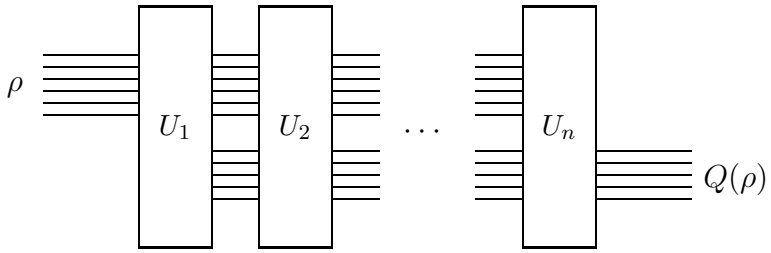


Figure 2: The original circuits Q_1 and Q_2 decomposed into constant depth unitary circuits.

and Q_2 first introduce any necessary ancillary qubits, then apply local unitary gates, and finally trace out any qubits that are not part of the input. This restriction can be made with no loss in generality, as any quantum circuit, even one that incorporates measurements and other non-unitary operations, can be approximated by such a circuit, and furthermore, this circuit uses a number of gates that is a polynomial in the size of the original circuit [1].

A simple way to decompose Q_1 into constant depth pieces is to simply let each gate of Q_1 be a piece in the decomposition. Let $U_1, U_2, \ldots, U_n$ be these pieces, with the additional complication that the operation U_1 both adds the ancillary qubits and performs the first gate of the circuit. In a similar way, Q_2 can be decomposed into constant depth pieces $V_1, V_2, \ldots, V_n$. These pieces are shown in Figure 2. If Q_1 and Q_2 implement transformations from $\mathcal{H}$ to $\mathcal{K}$, using ancillary qubits that fit into $\mathcal{A}$, and trace out the qubits in $\mathcal{B}$, then the spaces $\mathcal{H} \otimes \mathcal{A}$ and $\mathcal{B} \otimes \mathcal{K}$ are isomorphic, since by assumption Q_1 and Q_2 first introduce any needed ancilla and only trace qubits out at the end of the computation. Using these spaces, and implicitly this isomorphism, we have

$$\begin{aligned} U_1, V_1 &\in \mathbf{U}(\mathcal{H}_1, \mathcal{B}_1 \otimes \mathcal{K}_1) \\ U_i, V_i &\in \mathbf{U}(\mathcal{H}_i \otimes \mathcal{A}_i, \mathcal{B}_i \otimes \mathcal{K}_i) \quad \text{for } 2 \leq i \leq n, \end{aligned}$$

where the subscripted spaces are copies of the non-subscripted spaces that hold the input or output of one of the pieces of the original circuits. As an example of this notation, if $\rho \in \mathbf{D}(\mathcal{H})$, then the output of Q_1 on ρ is given by

$$\mathrm{tr}_{\mathcal{B}_n} U_n U_{n-1} \cdots U_1 \rho U_1^* U_2^* \cdots U_n^*,$$

and the output of Q_2 is given by the same expression using the V_i operators.

Using this decomposition of Q_1 and Q_2, circuits C_1 and C_2 are constructed that are logarithmic in depth and still in some sense faithfully implement Q_1 and Q_2. This is done by running the circuits corresponding to $U_1, \ldots, U_n$ in parallel, and tracing out all the qubits that are not in the output of U_n. Such a circuit is constant depth, but does not necessarily output a state in the image of Q_1, as the input to U_i is not necessarily close to the output from U_{i-1}. This problem can be dealt with by comparing the output of U_{i-1} to the input to U_i. In order to do this in logarithmic depth an auxiliary input that is first compared against the input to U_i and then held in reserve to compare to the output of U_{i-1} is needed. To compare these quantum states the swap test can be used. This test will fail with some probability depending on the distance between the two states. An example of the construction used to ensure that the output of U_{i-1} agrees with the input to U_i is given in Figure 3. To simplify the analysis of the constructed circuits these tests are controlled so that either one or the other is performed. This will affect the failure probability by a

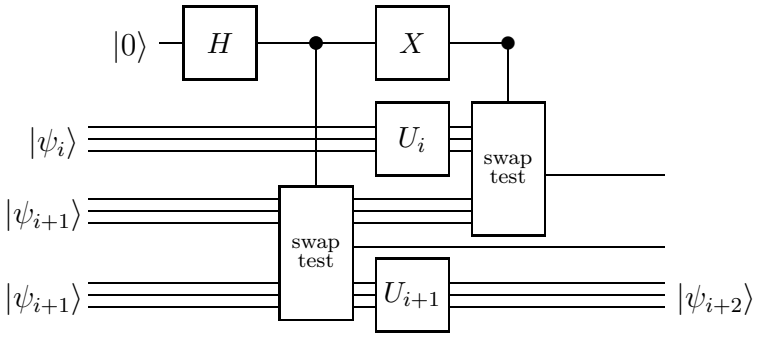


Figure 3: Testing that the output of U_i is close to the input of U_{i+1}. The inputs $|\psi_j\rangle$ are the ideal inputs to U_j, and are labelled for clarity only – no assumptions are made about these states. Qubits that do not reach the right edge are traced out.

factor of at most two, but will allow the analysis of each swap test to ignore the effect of the other. To implement this a control qubit is used so that either the first or the second test is performed between every two pieces U_i, U_{i+1} of the circuit. If a test is not performed, then the value of the output qubit of the swap test is left unchanged, and so the result of the test is a qubit in the $|0\rangle$ state. These controlled operations can be implemented in logarithmic depth using the technique of Moore and Nilsson [10].

After adding these tests between each piece of the circuit there is one final modification required. If any of the swap tests fail, i.e. detect states that are not the same, then they will output qubits in the $|1\rangle$ state. As yes instances of $\mathsf{CI}_{1,b}$ have outputs that are close together, we can ensure that no outputs of the constructed circuits can be close if any swap tests fail by adding dummy qubits in the $|0\rangle$ state to be compared to the outputs of the swap tests in the other circuit. These dummy qubits are shown in Figure 4.

The constructed circuits C_1 and C_2 are obtained by decomposing Q_1 and Q_2 into constant depth pieces, inserting the swap tests shown in Figure 3, and adding dummy qubits to ensure that the swap tests in the other circuit do not fail. At the end of these circuits, all qubits are traced out, except the output (in the space $\mathcal{K}_n$) of U_n or V_n, the output of the swap tests, and the dummy zero qubits. If the outputs of C_1 and C_2 are close together, then intuitively the output of the swap tests in each circuit must be close to zero and the output of U_n and V_n must also be close. If the swap tests do not fail with high probability (i.e. the outputs are close to zero), then these circuits will more or less faithfully reproduce the output of Q_1 and Q_2. Thus, in the case that the outputs of C_1 and C_2 can be made close, we will be able to argue that the output of Q_1 and Q_2 can also be made close. Proving that this intuitive picture is accurate forms the content of the next section.

In the other direction, it is not hard to see that if there are states $\rho, \xi \in \mathbf{D}(\mathcal{H})$ such that $Q_1(\rho) = Q_2(\xi)$, then there are similar states for the constructed circuits C_1 and C_2. To do this, notice that the circuit construction does not change if additional qubits are added to the circuits to allow purification of the states ρ and ξ to be used as inputs to C_1 and C_2. These additional qubits are traced out with the other qubits at the end of the circuit, so that the output state of the circuit are not changed. As these purifications are pure states and all operations performed during the circuit are unitary, the intermediate states of the

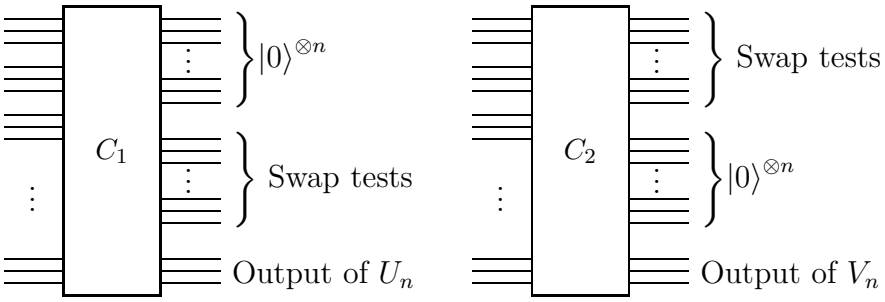


Figure 4: The outputs of C_1 and C_2.

circuits must also be pure states. If the input state to C_1 is $|\psi\rangle$, then by providing the state

$$|\psi\rangle \otimes U_1|\psi\rangle \otimes \cdots \otimes U_{n-1}U_{n-2}\cdots U_1|\psi\rangle$$

as input to C_1, the output of each block of the circuit will be identical to the input to the next block, ensuring that all the swap tests will succeed with probability one. It remains only to check on such input states that C_1 produces the same output as Q_1 on ρ. This can be observed by noting that the output of the circuit is exactly

$$\mathrm{tr}_{\mathcal{B}_n} U_n U_{n-1}\cdots U_1 \rho U_1^* U_2^* \cdots U_n^*,$$

which by construction is equal to the output of Q_1 on ρ. Thus if the circuits Q_1 and Q_2 have intersecting images then so do the circuits C_1 and C_2. This observation proves the completeness of the construction. Soundness is considerably more intricate, and is the focus of the next section.

5. Soundness of the Construction

In this section it is demonstrated that if the images of the original circuits Q_1 and Q_2 are far apart then so must be the images of the constructed circuits C_1 and C_2. As the constructed circuits essentially simulate Q_1 and Q_2 the desired result can be obtained by arguing that either the outputs of C_1 and C_2 are far apart or the input to at least one of the constructed circuits is not a faithful simulation of the corresponding original circuit. In the case that this simulation is not faithful it will be shown that there is some swap test that fails with reasonable probability. This implies that outputs of the constructed circuits must also be distant, as the failing swap test produces a state of the form $(1-p)|0\rangle\langle 0| + p|1\rangle\langle 1|$ that has low fidelity with the corresponding dummy zero qubit of the other circuit.

As a first step, we place a lower bound on the failure probability of a swap test in terms of the fidelity of the two states being compared. In the following lemma the swap test is viewed as a measurement of the symmetric and antisymmetric projectors, with the outcome that produces a qubit in the state $|1\rangle$ corresponding to the antisymmetric case.

Lemma 5.1. *If $\rho \in \mathbf{D}(\mathcal{A}\otimes\mathcal{B})$ then a swap test on $\mathcal{A}\otimes\mathcal{B}$ returns the antisymmetric outcome with probability at least*

$$\frac{1}{2} - \frac{1}{2}\,\mathrm{F}(\mathrm{tr}_{\mathcal{A}}\,\rho, \mathrm{tr}_{\mathcal{B}}\,\rho).$$

Proof. Let $|\psi\rangle \in \mathcal{A} \otimes \mathcal{B} \otimes \mathcal{C}$ be a purification of ρ, where $\mathcal{C}$ is an arbitrary space of sufficient dimension to allow such a purification. The swap test measures the state on $\mathcal{A} \otimes \mathcal{B}$ with the projectors $\frac{1}{2}(I - W)$ and $\frac{1}{2}(I + W)$, where W is the swap operator on $\mathcal{A} \otimes \mathcal{B}$. Thus, the antisymmetric outcome occurs with probability

$$\frac{1}{4} \operatorname{tr} \left([(I - W) \otimes I] |\psi\rangle\langle\psi| [(I - W^*) \otimes I] \right) = \frac{1}{2} \langle\psi| I \otimes I - W \otimes I |\psi\rangle = \frac{1}{2} - \frac{1}{2} \langle\psi| W \otimes I |\psi\rangle,$$

as W is hermitian. Then as W is also unitary, the states $|\psi\rangle$ and $W|\psi\rangle$ each purify both $\operatorname{tr}_{\mathcal{A}\otimes\mathcal{C}} |\psi\rangle\langle\psi|$ and $\operatorname{tr}_{\mathcal{B}\otimes\mathcal{C}} |\psi\rangle\langle\psi|$, and so by Uhlmann's theorem

$$\frac{1}{2} - \frac{1}{2} \langle\psi| W \otimes I |\psi\rangle \geq \frac{1}{2} - \frac{1}{2} \operatorname{F}(\operatorname{tr}_{\mathcal{A}\otimes\mathcal{C}} |\psi\rangle\langle\psi|, \operatorname{tr}_{\mathcal{B}\otimes\mathcal{C}} |\psi\rangle\langle\psi|).$$

After tracing out the space $\mathcal{C}$, this is exactly the statement of the lemma. ∎

This lemma cannot be immediately applied to the circuits C_1 and C_2, as in these circuits the output of one block of the circuit is not directly compared to the input to the next block, but instead each of these states are with probability 1/2 compared to some intermediate value. In order to deal with this difficulty, we use the Fuchs-van de Graaf inequalities to translate the fidelity to a relation involving the trace norm, which we can then apply the triangle inequality to. This application of the triangle inequality shows that at least one of the two swap tests fails with probability bounded below by an expression involving the fidelity. In the following corollary the reduced states of various parts of the input to either of the circuits C_1 or C_2 are used, but it is not assumed that these states are given in a separable form. For instance, the density matrices ρ_i, σ_i, and ξ_i that appear in the lemma may be part of some larger entangled pure state, so that the failure probabilities of the two swap tests need not be independent.

Corollary 5.2. *If $|\psi\rangle$ is input to the circuit C_a for $a \in \{1, 2\}$, with ρ_i the reduced state of $|\psi\rangle\langle\psi|$ on $\mathcal{H}_i \otimes \mathcal{A}_i$, then at least one of the swap tests on the ith block of C_a fails with probability at least*

$$\frac{1}{64} \left\| U_i \rho_{i-1} U_i^* - \rho_i \right\|_{\mathrm{tr}}^2 .$$

Proof. In the ith block of C_a there are two inputs to the first swap test: let the reduced density operators of these inputs be ρ_i and σ_i. The inputs to the second swap test are then given by σ_i and $U_i \rho_{i-1} U_i^* = \xi_i$. As exactly one of these tests is performed we do not need to consider the effect of the first test on the state when considering the second test, and so the same input state σ_i is used in both swap tests.

By Lemma 5.1, the failure probability of first and second tests, when performed, are at least $\frac{1}{2}(1 - \operatorname{F}(\rho_i, \sigma_i))$ and $\frac{1}{2}(1 - \operatorname{F}(\sigma_i, \xi_i))$, respectively. Thus, the probability p that at least one of these tests fails, given that each of them is performed with probability 1/2, is at least

$$p \geq \frac{1}{2} \max \left\{ \frac{1}{2}(1 - \operatorname{F}(\sigma_i, \xi_i)), \frac{1}{2}(1 - \operatorname{F}(\rho_i, \sigma_i)) \right\} = \frac{1}{4} \left(1 - \min\{\operatorname{F}(\sigma_i, \xi_i)), \operatorname{F}(\rho_i, \sigma_i)\} \right).$$

By the Fuchs-van de Graaf inequalities, this fidelity may be replaced by the trace norm. Doing so, we obtain

$$p \geq \frac{1}{16} \max(\left\| \sigma_i - \xi_i \right\|_{\mathrm{tr}}^2 , \left\| \rho_i - \sigma_i \right\|_{\mathrm{tr}}^2).$$

Finally, as this maximum must be at least the average of the two values,

$$p \geq \frac{1}{16}\left(\frac{\|\sigma_i - \xi_i\|_{\mathrm{tr}}}{2} + \frac{\|\rho_i - \sigma_i\|_{\mathrm{tr}}}{2}\right)^2 \geq \frac{1}{64}\|\rho_i - \xi_i\|_{\mathrm{tr}}^2,$$

where the last inequality follows from an application of the triangle inequality. ∎

By repeatedly applying some of the properties of the trace norm discussed in Section 2 it is somewhat tedious but not difficult to reduce the problem at hand to the previous Corollary. This is the content of the following theorem.

Theorem 5.3. *If* $\mathrm{F}(Q_1(\rho_0), Q_2(\xi_0)) < 1 - c$ *for all* $\rho_0, \xi_0 \in \mathcal{H}$ *then*

$$\mathrm{F}(C_1(\rho), C_2(\xi)) < 1 - \frac{c^2}{144n^2}$$

for all $\rho, \xi \in (\mathcal{H} \otimes \mathcal{A})^{\otimes 2n}$.

Proof. Let ρ and ξ be inputs to C_1 and C_2, and let ρ_i, ξ_i be the reduced states of these inputs on $\mathcal{H}_i \otimes \mathcal{A}_i$ for $0 \leq i \leq 2n$, where the states for $i > n$ are the inputs that are only used by the swap tests, which we will not need to refer to explicitly. That is, ρ_i and ξ_i for $0 \leq i \leq n$ are the portions of the state that are input to the unitaries U_i and V_i that make up the circuits Q_1 and Q_2. The output of the circuits C_1 and C_2 is then given by a number of qubits corresponding to the swap tests as well as the states $\mathrm{tr}_{\mathcal{B}_n} \rho_n$ and $\mathrm{tr}_{\mathcal{B}_n} \xi_n$, where $\mathcal{B}_n$ is simply the space that is traced out to obtain the output from the unitary representations of the original circuits.

By the condition on the fidelity of Q_1 and Q_2 and the Fuchs-van de Graaf inequalities, we have $2c < \|Q_1(\rho_0) - Q_2(\xi_0)\|_{\mathrm{tr}}$. Using the triangle inequality we can relate this to the distance between the constructed circuits. Adding terms and simplifying, we obtain

$$\begin{aligned} 2c &< \|Q_1(\rho_0) - \mathrm{tr}_{\mathcal{B}_n} \rho_n + \mathrm{tr}_{\mathcal{B}_n} \xi_n - Q_2(\xi_0) + \mathrm{tr}_{\mathcal{B}_n} \rho_n - \mathrm{tr}_{\mathcal{B}_n} \xi_n\|_{\mathrm{tr}} \\ &\leq \|Q_1(\rho_0) - \mathrm{tr}_{\mathcal{B}_n} \rho_n\|_{\mathrm{tr}} + \|\mathrm{tr}_{\mathcal{B}_n} \xi_n - Q_2(\xi_0)\|_{\mathrm{tr}} + \|\mathrm{tr}_{\mathcal{B}_n} \rho_n - \mathrm{tr}_{\mathcal{B}_n} \xi_n\|_{\mathrm{tr}}. \end{aligned}$$

We now observe that $\|\mathrm{tr}_{\mathcal{B}_n} \rho_n - \mathrm{tr}_{\mathcal{B}_n} \xi_n\|_{\mathrm{tr}} \leq \|C_1(\rho) - C_2(\xi)\|_{\mathrm{tr}}$ by the monotonicity of the trace norm under the partial trace, since the former can be obtained from the later by tracing out the appropriate spaces. Using this we have

$$2c < \|Q_1(\rho_0) - \mathrm{tr}_{\mathcal{B}_n} \rho_n\|_{\mathrm{tr}} + \|\mathrm{tr}_{\mathcal{B}_n} \xi_n - Q_2(\xi_0)\|_{\mathrm{tr}} + \|C_1(\rho) - C_2(\xi)\|_{\mathrm{tr}} \tag{5.1}$$

As the three terms on the right are nonnegative, at least one of them must be larger than the average $2c/3$. If $\|C_1(\rho) - C_2(\xi)\|_{\mathrm{tr}} > 2c/3$ then $\mathrm{F}(C_1(\rho), C_2(\xi)) < 1 - c^2/144$ and there is nothing left to prove.

The cases where one of the first two terms of (5.1) exceeds $2c/3$ are symmetric, and so we can consider only the first term. Expanding $Q_1(\rho_0)$ in terms of the U_i, we obtain

$$\begin{aligned} \frac{2c}{3} &< \|Q_1(\rho_0) - \mathrm{tr}_{\mathcal{B}_n} \rho_n\|_{\mathrm{tr}} \\ &= \|\mathrm{tr}_{\mathcal{B}_n} U_n U_{n-1} \cdots U_1 \rho_0 U_1^* U_2^* \cdots U_n^* - \mathrm{tr}_{\mathcal{B}_n} \rho_n\|_{\mathrm{tr}} \\ &\leq \|U_n U_{n-1} \cdots U_1 \rho_0 U_1^* U_2^* \cdots U_n^* - \rho_n\|_{\mathrm{tr}}, \end{aligned}$$

where once again the monotonicity of the trace norm under the partial trace has been used. By repeating the strategy of adding terms and then applying the triangle inequality we have

$$\frac{2c}{3} < \|U_1 \rho_0 U_1^* - \rho_1\|_{\mathrm{tr}} + \|U_n U_{n-1} \cdots U_2 \rho_1 U_2^* U_3^* \cdots U_n^* - \rho_n\|_{\mathrm{tr}}.$$

Here we have made use of the unitary invariance of the trace norm to discard the operators $U_2, \ldots U_n$ from the first term. Continuing in this fashion we have

$$\frac{2c}{3} < \sum_{i=1}^{n} \|U_i \rho_{i-1} U_i^* - \rho_i\|_{\mathrm{tr}} .$$

As all terms in this sum are nonnegative, there must be at least one term in the sum that exceeds $2c/(3n)$, as this is a lower bound on the average of all terms. Thus, for some value of i, we have $\|U_i \rho_{i-1} U_i^* - \rho_i\|_{\mathrm{tr}} > 2c/(3n)$, and so by Corollary 5.2 one of the corresponding swap tests fails with probability $p > c^2/(144n^2)$. The qubit representing the output value of this swap test is then of the form $(1-p)|0\rangle\langle 0| + p|1\rangle\langle 1|$, and so, by the monotonicity of the fidelity under the partial trace,

$$\mathrm{F}(C_1(\rho), C_2(\xi)) \leq \mathrm{F}((1-p)|0\rangle\langle 0| + p|1\rangle\langle 1|, |0\rangle\langle 0|) = 1 - p < 1 - \frac{c^2}{144n^2},$$

as in the statement of the theorem. ∎

By combining Theorem 5.3 with the observation in Section 4 and the multiplicativity of the maximum output fidelity of two transformations, we obtain the following result.

Corollary 5.4. *The problem* Log-depth $\mathsf{CI}_{1,b}$ *is* QIP*-complete for any constant* $0 < b < 1$.

Proof. Theorem 5.3 establishes the completeness of the problem for any $b \geq 1 - c^2/(144n^2)$, where n is an upper bound on the size of the circuits. Using Theorem 2.1 of Kitaev, Shen, and Vyalyi [9] we can repeat each of the circuits r times in parallel to obtain the completeness of the problem for $b \geq \left(1 - c^2/(144n^2)\right)^r$, which can be made smaller than any constant for r some polynomial in n. ∎

As the circuits constructed by the reduction only make use of logarithmic depth when performing swap tests, and the controlled swap operations performed by these tests can be accomplished in constant depth using unbounded fan-out gates, the following Corollary follows immediately from the previous one.

Corollary 5.5. *The problem* Const-depth $\mathsf{CI}_{1,b}$ *on circuits with the unbounded fan-out gate is* QIP*-complete for for any constant* $0 < b < 1$.

6. Distinguishing Log-Depth Computations

The hardness of Log-depth $\mathsf{CI}_{1,b}$ can be extended to Log-depth $\mathsf{QCD}_{2,b}$ by observing that the reduction for the polynomial depth version of this problem in [12] can be made to preserve the depth of the constructed circuits. Once this observation is made, the hardness of the log-depth (and constant-depth with fan-out) versions of the circuit distinguishability problem is immediate.

The reduction in [12] takes as input circuits (Q_1, Q_2) and produces circuits C_1 and C_2. Without describing the reduction in detail, the constructed circuits C_1 and C_2 run, depending on the value of a control qubit, one of Q_1 and Q_2, followed by a constant depth circuit. If the input circuits Q_1 and Q_2 have logarithmic depth, then the only significant difficulty is the fact that controlled versions of these circuits are needed. However, as we have already seen, if we replace the gates in Q_1 and Q_2 with controlled versions, then we can use the scheme of Moore and Nilsson [10] to implement the controlled operations in

logarithmic depth. With this modification, the reduction in [12] can be reused to show the hardness of the QCD problem on log-depth circuits.

Corollary 6.1. Log-depth $\mathsf{QCD}_{2,b}$ *is* QIP*-complete for any constant* $0 < b < 2$.

Once again these controlled operations can be implemented in a constant depth circuit if the unbounded fan-out gate is allowed into the set of allowed gates.

Corollary 6.2. Const-depth $\mathsf{QCD}_{2,b}$ *on circuits with the unbounded fan-out gate is* QIP*-complete for any constant* $0 < b < 2$.

7. Conclusion

The hardness of distinguishing even log-depth mixed state quantum circuits leaves several related open problems, a few of which are listed here.

- Can this new complete problem be used to further understand QIP?
- Does this result rely in an essential way on the mixed state circuit model? How difficult is it to distinguish quantum circuits in less general models of computation?
- What is the complexity of distinguishing constant depth quantum circuits that do not use the unbounded fan-out gate?

Acknowledgement. I would like to thank John Watrous for several helpful discussions, the anonymous reviewers for constructive comments, as well as Canada's NSERC and MITACS for supporting this research.

References

[1] D. Aharonov, A. Kitaev, N. Nisan. Quantum circuits with mixed states. In *Proc. 30th ACM Symposium on the Theory of Computing*, pp. 20–30, 1998.

[2] A. Broadbent, E. Kashefi. Parallelizing quantum circuits. arXiv:0704.1736v1 [quant-ph].

[3] R. Cleve, J. Watrous. Fast parallel circuits for the quantum fourier transform. In *Proc. 41st ACM Symposium on the Theory of Computing*, pp. 526–536, 2000.

[4] S. Fenner, F. Green, S. Homer, Y. Zhang. Bounds on the power of constant-depth quantum circuits. In *Proc. 15th International Symposium on Fundamentals of Computation Theory*, pp. 44–55, 2005.

[5] C. A. Fuchs, J. van de Graaf. Cryptographic distinguishability measures for quantum-mechanical states. *IEEE Transactions on Information Theory*, 45(4):1216–1227, 1999.

[6] F. Green, S. Homer, C. Moore, C. Pollett. Counting, fanout and the complexity of quantum ACC. *Quantum Information and Computation*, 2(1):35–65, 2002.

[7] P. Høyer, R. Špalek. Quantum circuits with unbounded fan-out. *Theory of Computing*, 1:81–103, 2005.

[8] A. Kitaev, J. Watrous. Parallelization, amplification and exponential time simulation of quantum interactive proof systems. In *Proc. 32nd ACM Symp. Theory of Computing*, pp. 608–617, 2000.

[9] A. Y. Kitaev, A. H. Shen, M. N. Vyalyi. *Classical and Quantum Computation*, volume 47 of *Graduate Studies in Mathematics*. American Mathematical Society, 2002.

[10] C. Moore, M. Nilsson. Parallel quantum computation and quantum codes. *SIAM Journal on Computing*, 31(3):799–815, 2002.

[11] M. A. Nielsen, I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[12] B. Rosgen, J. Watrous. On the hardness of distinguishing mixed-state quantum computations. In *Proc. 20th Conference on Computational Complexity*, pp. 344–354, 2005.

[13] B. M. Terhal, D. P. DiVincenzo. Adaptive quantum computation, constant depth quantum circuits and Arthur-Merlin games. *Quantum Information and Computation*, 4(2):134–145, 2004.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 609-620
www.stacs-conf.org

FACTORING POLYNOMIALS OVER FINITE FIELDS USING BALANCE TEST

CHANDAN SAHA

Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur
E-mail address: csaha@cse.iitk.ac.in

Abstract. We study the problem of factoring univariate polynomials over finite fields. Under the assumption of the Extended Riemann Hypothesis (ERH), Gao [Gao01] designed a polynomial time algorithm that fails to factor only if the input polynomial satisfies a strong symmetry property, namely *square balance*. In this paper, we propose an extension of Gao's algorithm that fails only under an even stronger symmetry property. We also show that our property can be used to improve the time complexity of best deterministic algorithms on most input polynomials. The property also yields a new randomized polynomial time algorithm.

1. Introduction

We consider the problem of designing an efficient deterministic algorithm for factoring a univariate polynomial, with coefficients taken from a finite field. The problem reduces in polynomial time to the problem of factoring a monic, square-free and completely splitting polynomial $f(x)$ with coefficients in a prime field F_p (see [Ber70], [LN94]). Although there are efficient polynomial time randomized algorithms for factoring $f(x)$ ([Ber70], [CZ81], [vzGS92], [KS95]), as yet there is no deterministic polynomial time algorithm even under the assumption of the Extended Riemann Hypothesis (ERH). In this paper we will assume that ERH is true and $\xi_1, \xi_2, \ldots, \xi_n$ are the n distinct roots of the input polynomial f,

$$f(x) = \prod_{i=1}^{n} (x - \xi_i) \quad \text{where } \xi_i \in F_p$$

In 2001, Gao [Gao01] gave a deterministic factoring algorithm that fails to find non-trivial factors of f in polynomial time, if f belongs to a restricted class of polynomials, namely *square balanced polynomials*. Motivated by the work of Gao [Gao01], we have defined a proper subclass of square balanced polynomials, namely *cross balanced polynomials*, such that polynomials that are not cross balanced, can be factored deterministically in polynomial time, under the assumption of the ERH.

Our contribution can be summarized as follows. Let f be a monic, square-free and completely splitting polynomial in $F_p[x]$ with n roots $\xi_1, \ldots, \xi_n$. Our factoring algorithm uses an arbitrary (but deterministically chosen) collection of $k = (n \log p)^{O(1)}$ $(n = deg(f))$

Key words and phrases: Algebraic Algorithms, polynomial factorization, finite fields.

small degree auxiliary polynomials $p_1(.), \ldots, p_k(.)$, and from each $p_l(\cdot)$ $(1 \leq l \leq k)$ and f it implicitly constructs a simple n-vertex digraph G_l such that, (for $l > 1$) G_l is a subgraph (not necessarily a proper subgraph) of G_{l-1}. A proper factor of f is efficiently retrieved if any one of the graphs is either not regular, or is regular with in degree and out degree of every vertex less than a chosen constant c. This condition of regularity of all the k graphs imposes a *tight symmetry condition* on the roots of f, and we point out that this may be exploited to improve the worst case time complexity of the best known deterministic algorithms. Further, we show that if the polynomials $p_l(\cdot)$ $(1 \leq l \leq k)$ are randomly chosen then the symmetry breaks with high probability and our algorithm works in randomized polynomial time. We call the checking of this symmetry condition a *balance test*.

We now present a little more details. Define the sets Δ_i for $1 \leq i \leq n$ as,

$$\Delta_i = \{1 \leq j \leq n : j \neq i, \sigma((\xi_i - \xi_j)^2) = -(\xi_i - \xi_j)\}$$

where σ is the square root algorithm described in [Gao01] (see section 2.4). The polynomial f is called a *square balanced polynomial* (as in [Gao01]) if $\#\Delta_1 = \ldots = \#\Delta_n$. For $l > 1$, define polynomial f_l as,

$$f_l = \prod_{i=1}^{n} (x - p_l(\xi_i))$$

where $p_l(.)$ is an arbitrary but deterministically chosen polynomial with degree bounded by $(n \log p)^{O(1)}$. Further, $p_{l_1}(.) \neq p_{l_2}(.)$ for $l_1 \neq l_2$, and f_1 is taken to be f i.e. $p_1(y) = y$. Assume that, for a given $k = (n \log p)^{O(1)}$, for every l, $1 \leq l \leq k$, polynomial $f_l = \tilde{f}_l^{d_l}$, where $\tilde{f}_l$ is a square-free and square balanced polynomial and $d_l > 0$. Later, we show that, if f_l is not of the above form then a proper factor of f can be retrieved efficiently. For each polynomial f_l, $1 \leq l \leq k$, define the sets $\Delta_i^{(l)}$ for $1 \leq i \leq n$ as,

$$\Delta_i^{(l)} = \{1 \leq j \leq n : p_l(\xi_i) \neq p_l(\xi_j), \sigma((p_l(\xi_i) - p_l(\xi_j))^2) = -(p_l(\xi_i) - p_l(\xi_j))\}$$

Further, define the sets ${D_i}^{(l)}$ iteratively over l as,

$$\begin{aligned} D_i^{(1)} &= \Delta_i^{(1)} \\ \text{For } l > 1,\ D_i^{(l)} &= D_i^{(l-1)} \cap \Delta_i^{(l)} \\ \text{If } D_i^{(l)} &= \phi \text{ for all } i,\ 1 \leq i \leq n, \text{ then redefine } D_i^{(l)} \text{ as } D_i^{(l)} = D_i^{(l-1)}. \end{aligned}$$

For $1 \leq l \leq k$, let G_l be a directed graph with n vertices $v_1, \ldots, v_n$, such that there is an edge from v_i to v_j if and only if $j \in D_i^{(l)}$. Note that, G_l is a subgraph of G_{l-1} for $1 < l \leq k$. Denote the in degree and out degree of a vertex v_i by $indeg(v_i)$ and $outdeg(v_i)$, respectively. We say that the graph G_l is *regular* (or *t-regular*) if $indeg(v_1) = outdeg(v_1) = \ldots = indeg(v_n) = outdeg(v_n) = t$. Call t as the *regularity* of G_l. The following theorem is proved in this paper.

Theorem 1.1. *Polynomial f can be factored into nontrivial factors in time $l \cdot (n \log p)^{O(1)}$ if G_l is not regular for some l, $1 \leq l \leq k$. Further, if $G_1, \ldots, G_k$ are all regular and for at least $\lceil \log_2 n \rceil$ of the graphs we have $G_l \neq G_{l-1}$ $(1 < l \leq k)$, then f can be factored in $k \cdot (n \log p)^{O(1)}$ time.*

Note that, G_1 is regular if and only if f is square balanced, as $\Delta_i^{(1)} = \Delta_i$, for $1 \leq i \leq n$ and G_1 is in fact a regular tournament.

Suppose $f(y)$ splits as $f(y) = (y - X) \cdot f'(y)$ in the quotient ring $R = \frac{F_p[x]}{(f)}$ where $X = x \mod f$. Our algorithm iteratively tests graphs $G_1, G_2, \ldots$ so on, to check if any one of them is not regular. If at the l^{th} iteration graph G_l turns out to be not regular, then a proper factor of f is obtained in polynomial time. However, if G_l is regular, then the algorithm returns a nontrivial monic factor $g_l(y)$ of $f'(y)$ with degree equal to the regularity of G_l. Moreover, $g_l(y)$ is also a factor of (although may be equal to) $g_{l-1}(y)$, the factor obtained at the $(l-1)^{th}$ iteration, and it can be ensured that if $g_l(y)$ is a proper factor of $g_{l-1}(y)$ (which happens iff $G_l \neq G_{l-1}$) then $deg(g_l(y)) \leq \frac{1}{2} \cdot deg(g_{l-1}(y))$. Thus, if the graphs repeatedly turn out to be regular (which in itself is a stringent condition) and for at least $\lceil \log_2 n \rceil$ times it happen that $G_l \neq G_{l-1}$, for $1 < l \leq k$, then we obtain a nontrivial linear factor $g(y)$ of $f'(y)$. The element $-g(0)$ defines a nontrivial *endomorphism* in the ring R, and by using a result from [Evd94] (Lemma 9 in [Evd94]) we can find a proper factor of f in polynomial time. Further, if for only $\epsilon \lceil \log_2 n \rceil$ times we get $G_l \neq G_{l-1}$ $(1 < l \leq k)$ for some ϵ, $0 < \epsilon \leq 1$, then we obtain a nontrivial factor $g(y)$ of $f'(y)$ with degree at most $\frac{n^{1-\epsilon}}{2}$. Now if we apply Evdokimov's algorithm ([Evd94]) on $g(y)$ (instead of $f'(y)$), we can get a proper factor of f in time $(n^{\frac{(1-\epsilon)^2}{2} \log n + \epsilon + c_1} \log p)^{c_2}$ (c_1 and c_2 are constants). For most polynomials $\epsilon > 0$ (i.e. at least about $\frac{1}{\log n}$) and this gives an improvement over the time complexity of $(n^{\frac{1}{2} \log n + c_1} \log p)^{c_2}$ in [Evd94] (c_1, c_2 are the same constants).

Assuming $n << p$, all the best known deterministic algorithms (e.g. [Evd94], [CH00]) use computations in rings with large dimensions over F_p to get smaller degree factors of $f'(y)$. Unlike these approaches, the balance test is an attempt to exploit an asymmetry among the roots of the input polynomial to obtain smaller degree factors of $f'(y)$ without carrying out computations in rings with large dimensions over F_p. This attribute of our approach yields a better time complexity for most polynomials in a way as discussed in the previous paragraph.

It is sufficient to choose the auxiliary polynomials $p_l(y)$, $1 < l \leq k$, in such a way that the graphs, if regular, are not all the same for too long, if their regularities are large. An efficient and deterministic construction of such auxiliary polynomials will immediately imply that factorization of univariate polynomials over finite fields can be done in deterministic polynomial time under ERH. In this paper we assume that the auxiliary polynomials are arbitrary but deterministically chosen polynomials with degree bounded by $(n \log p)^{O(1)}$. For example, one possibility is to choose $p_l(y) = y^l$ for $1 \leq l \leq k$. (In fact, Gao [Gao01] used this choice of auxiliary polynomials to define a restricted class of square balanced polynomials called *super square balanced polynomials*.) We show that, if random choices of auxiliary polynomials are allowed then our algorithm works in randomized polynomial time. For the graphs to be all regular and equal, the roots of f must satisfy a tight symmetry condition (given by equal sizes of all the sets $D_i^{(l)}$, for $1 \leq i \leq n$ and $1 \leq l \leq k$) and it is only then that our algorithm fails to factor f.

Definition 1.1. A polynomial f is called *k-cross balanced*, for $k > 0$, if for every l, $1 \leq l \leq k$, polynomial $f_l = \tilde{f}_l^{d_l}$, where $\tilde{f}_l$ is a square-free, square balanced polynomial with $d_l > 0$, and graph G_l is regular.

It follows from the definition that, 1-cross balanced polynomials form the class of square balanced polynomials. Let $k = (n \log p)^{O(1)}$ be some fixed polynomial in n and $\log p$. A polynomial f is called *cross balanced* if it is k-cross balanced and regularity of graph G_k is

greater than a fixed constant c. From Theorem 1.1 and [Evd94] it follows that, polynomials that are not cross balanced can be factored deterministically in polynomial time.

2. Preliminaries

Assume that f is a monic, square-free and completely splitting polynomial over F_p and $R = \frac{F_p[x]}{(f)}$ is the quotient ring consisting of all polynomials modulo f.

2.1. Primitive Idempotents

Elements $\chi_1, \ldots, \chi_n$ of the ring R are called the *primitive idempotents* of R if, $\sum_{i=1}^{n} \chi_i = 1$ and for $1 \leq i, j \leq n$, $\chi_i \cdot \chi_j = \chi_i$ if $i = j$ and 0 otherwise. By Chinese Remaindering theorem, $R \cong F_p \oplus \ldots \oplus F_p$ (n times), such that every element in R can be uniquely represented by an n-tuple of elements in F_p. Addition and multiplication between two elements in R can viewed as componentwise addition and multiplication of the n-tuples. Any element $\alpha = (a_1, \ldots, a_n) \in R$ can be equated as, $\alpha = \sum_{i=1}^{n} a_i \chi_i$ where $a_i \in F_p$. Let $g(y)$ be a polynomial in $R[y]$ given by,

$$\begin{aligned} g(y) &= \sum_{i=0}^{m} \gamma_i y^i \quad \text{where } \gamma_i \in R \text{ and} \\ \gamma_i &= \sum_{j=1}^{n} g_{ij} \chi_j \quad \text{where } g_{ij} \in F_p \text{ for } 0 \leq i \leq m \text{ and } 1 \leq j \leq n. \end{aligned}$$

Then $g(y)$ can be alternatively represented as,

$$g(y) = \sum_{j=1}^{n} g_j(y) \chi_j \text{ where } g_j(y) = \sum_{i=0}^{m} g_{ij} y^i \in F_p[y] \text{ for } 1 \leq j \leq n.$$

The usefulness of this representation is that, operations on polynomials in $R[y]$ (multiplication, gcd etc.) can be viewed as componentwise operations on polynomials in $F_p[y]$.

2.2. Characteristic Polynomial

Consider an element $\alpha = \sum_{i=1}^{n} a_i \chi_i \in R$ where $a_i \in F_p$, $1 \leq i \leq n$. The element α defines a linear transformation on the vector space R (over F_p), mapping an element $\beta \in R$ to $\alpha\beta \in R$. The characteristic polynomial of α (viewed as a linear transformation) is independent of the choice of basis and is equal to

$$c_\alpha(y) = \prod_{i=1}^{n} (y - a_i),$$

In order to construct c_α one can use $1, X, X^2, \ldots, X^{n-1}$ as the basis in R and form the matrix (m_{ij}) where $\alpha \cdot X^{j-1} = \sum_{i=1}^{n} m_{ij} X^{i-1}$, $m_{ij} \in F_p$, $1 \leq i, j \leq n$. Then c_α can be constructed by evaluating $det(y \cdot I - (m_{ij}))$ at n distinct values of y and solving for the n coefficients of c_α using linear algebra. The process takes only polynomial time. The notion of characteristic polynomial extends even to higher dimensional algebras over F_p.

2.3. GCD of Polynomials

Let $g(y) = \sum_{i=1}^{n} g_i(y)\chi_i$ and $h(y) = \sum_{i=1}^{n} h_i(y)\chi_i$ be two polynomials in $R[y]$, where $g_i, h_i \in F_p[y]$ for $1 \leq i \leq n$. Then, *gcd* of g and f is defined as,

$$gcd(g, f) = \sum_{i=1}^{n} gcd(g_i, h_i)\chi_i$$

We note that, the concept of *gcd* of polynomials does not make sense in general over any arbitrary algebra. However, the fact that R is a *completely splitting semisimple algebra* over F_p allows us to work component-wise over F_p and this makes the notion of *gcd* meaningful in the context. The following lemma was shown by Gao [Gao01].

Lemma 2.1. [Gao01] *Given two polynomials $g, h \in R[y]$, $gcd(g, h)$ can be computed in time polynomial in the degrees of the polynomials, n and $\log p$.*

2.4. Gao's Algorithm

Let $R = \frac{F_p[x]}{(f)} = F_p[X]$ where $X = x \mod f$ and suppose that $f(y)$ splits in R as, $f(y) = (y - X)f'(y)$. Define quotient ring S as, $S = \frac{R[y]}{(f')} = R[Y]$ where $Y = y \mod f'$. S is an elementary algebra over F_p with dimension $n' = n(n-1)$. Gao [Gao01] described an algorithm σ for taking square root of an element in S. If $p - 1 = 2^e w$ where $e \geq 1$ and w is odd, and η is a primitive 2^e-th root of unity, then σ has the following properties:

(1) Let $\mu_1, \ldots, \mu_{n'}$ be primitive idempotents in S and $\alpha = \sum_{i=1}^{n'} a_i\mu_i \in S$ where $a_i \in F_p$. Then, $\sigma(\alpha) = \sum_{i=1}^{n'} \sigma(a_i)\mu_i$.
(2) Let $a = \eta^u\theta$ where $\theta \in F_p$ with $\theta^w = 1$ and $0 \leq u < 2^e$. Then $\sigma(a^2) = a$ iff $u < 2^{e-1}$.

When $p = 3 \mod 4$, $\eta = -1$ and property 2 implies that $\sigma(a^2) = a$ for $a \in F_p$ iff a is a quadratic residue in F_p.

Algorithm 1. [Gao01]
Input: A polynomial $f \in F_p[x]$.
Output: A proper factor of f or output that "f is square balanced".
1. Form X, Y, R, S as before.
2. Compute $C = \frac{1}{2}(X + Y + \sigma((X - Y)^2)) \in S$.
3. Compute the characteristic polynomial $c(y)$ of C over R.
4. Decompose $c(y)$ as $c(y) = h(y)(y - X)^t$, where t is the largest possible.
5. If $h(X)$ is a zero divisor in R then find a proper factor of f, otherwise output that "f is square balanced".

It was shown in [Gao01] that Algorithm 1 fails to find a proper factor of f if and only if f is square balanced. Moreover, it follows from the analysis in [Gao01] (see Theorem 3.1 in [Gao01]) that, when f is square balanced the polynomial $h(y)$ takes the form,

$$h(y) = \sum_{i=1}^{n} \left[\prod_{j \in \Delta_i} (y - \xi_j) \right] \chi_i$$

where $\Delta_i = \{j : j \neq i, \sigma((\xi_i - \xi_j)^2) = -(\xi_i - \xi_j)\}$ and $\#\Delta_i = \frac{n-1}{2}$ for all i, $1 \leq i \leq n$.

3. Our Algorithm and Analysis

In this section, we describe our algorithm for factoring polynomial f. We show that the algorithm fails to factor f in $k \cdot (n \log p)^{O(1)}$ time if and only if f is k-cross balanced and regularity of G_k is greater than c. The algorithm involves k polynomials, $f = f_1, \ldots, f_k$, where polynomial f_l, $1 < l \leq k$, is defined as,

$$f_l = \prod_{i=1}^{n} (x - p_l(\xi_i))$$

where $p_l(.)$ is an arbitrary but deterministically fixed polynomial with degree bounded by $(n \log p)^{O(1)}$ and $p_{l_1}(.) \neq p_{l_2}(.)$ for $l_1 \neq l_2$. The polynomial f_l can be constructed in polynomial time by considering the element $p_l(X)$ in $R = \frac{F_p[x]}{(f)} = F_p[X]$, where $X = x \mod f$, and then computing its characteristic polynomial over F_p.

Lemma 3.1. *If f_l is not of the form $f_l = \tilde{f}_l^{\,d_l}$, where $\tilde{f}_l$ is a square-free, square balanced polynomial and $d_l > 0$, then a proper factor of f can be retrieved in polynomial time.*

Proof: By definition, $f_l = \prod_{i=1}^{n} (x - p_l(\xi_i))$. Define the sets E_i, for $1 \leq i \leq n$, as $E_i = \{1 \leq j \leq n : p_l(\xi_j) = p_l(\xi_i)\}$. Consider the following *gcd* in the ring $R[y]$,

$$g(y) = gcd\,(p_l(y) - p_l(X), f(y)) = \sum_{i=1}^{n} \left[\prod_{j \in E_i} (y - \xi_j) \right] \chi_i$$

The leading coefficient of $g(y)$ is a zero-divisor in R, unless $\#E_1 = \ldots = \#E_n = d_l$ (say). Therefore, we can assume that,

$$\begin{aligned} f_l &= \prod_{j=1}^{m_l} \left(x - p_l(\xi_{s_j})\right)^{d_l} \quad \text{where } p_l(\xi_{s_1}), \ldots, p_l(\xi_{s_{m_l}}) \text{ are all distinct and } m_l = \frac{n}{d_l} \\ &= \tilde{f}_l^{\,d_l} \quad \text{where } \tilde{f}_l = \prod_{j=1}^{m_l} \left(x - p_l(\xi_{s_j})\right) \text{ is square-free.} \end{aligned}$$

If polynomial $\tilde{f}_l$ (obtained by square-freeing f_l) is not square balanced then a proper factor $\tilde{g}_l$ of $\tilde{f}_l$ is returned by Algorithm 1. But then,

$$gcd\,(\tilde{g}_l(p_l(x)), f(x)) = \prod_{j : \tilde{g}_l(p_l(\xi_j)) = 0} (x - \xi_j)$$

is a proper factor of f. ∎

Algorithm 1 works with $\tilde{f}_l = \prod_{j=1}^{m_l} \left(x - p_l(\xi_{s_j})\right)$ as the input polynomial where $p_l(\xi_{s_j})$'s are distinct and $m_l = \frac{n}{d_l}$, and returns a polynomial $h_l(y)$ such that,

$$h_l(y) = \sum_{j=1}^{m_l} \left[\prod_{r \in \tilde{\Delta}_j^{(l)}} (y - p_l(\xi_{s_r})) \right] \chi_j^{(l)} \tag{3.1}$$

where $\chi_j^{(l)}$'s are the primitive idempotents of the ring $R_l = \frac{F_p[x]}{(\tilde{f}_l)}$,

$$\tilde{\Delta}_j^{(l)} = \{1 \leq r \leq m_l : r \neq j, \sigma((p_l(\xi_{s_j}) - p_l(\xi_{s_r}))^2) = -(p_l(\xi_{s_j}) - p_l(\xi_{s_r}))\}$$

and $\#\tilde{\Delta}_j^{(l)} = \frac{m_l-1}{2}$ for $1 \leq j \leq m_l$. Assume that $p > n^2$ and n is odd, as even degree polynomials can be factored in polynomial time. In the following algorithm, parameter k is taken to be a fixed polynomial in n and $\log p$ and c is a fixed constant.

Algorithm 2. Cross Balance
Input: A polynomial $f \in F_p[x]$ of odd degree n.
Output: A proper factor of f or "Failure".

- Choose $k-1$ distinct polynomials $p_2(y), \ldots, p_k(y)$ with degree greater than unity and bounded by a polynomial in n and $\log p$. (We can use any arbitrary, efficient mechanism to deterministically choose the polynomials.) Take $p_1(y) = y$.
- **for** $l = 1$ **to** k **do**

  [Steps (1) - (2): Constructing polynomial f_l and checking if f can be factored using Lemma 3.1.]

  (1) (*Construct polynomial* f_l) Compute the characteristic polynomial, $c_\alpha(x)$, of element $\alpha = p_l(X) \in R$, over F_p. Then $f_l = c_\alpha(x)$.
  (2) (*Check if* f *can be factored*) Check if f_l is of the form $f_l = \tilde{f}_l^{\,d_l}$, where $\tilde{f}_l$ is a square-free, square balanced polynomial and $d_l > 0$. If not, then find a proper factor of f as in Lemma 3.1.

  [Steps (3) - (6): Constructing graph G_l implicitly.]

  (3) (*Obtain the required polynomial from Algorithm* 1) Else, $\tilde{f}_l$ is square balanced and Algorithm 1 returns a polynomial $h_l(y) = y^t + \alpha_1 y^{t-1} + \ldots + \alpha_t$ (as in equation 3.1), where $t = \frac{m_l-1}{2}$ and $\alpha_u \in R_l$ for $1 \leq u \leq t$.
  (4) (*Change to a common ring so that gcd is feasible*) Each $\alpha_u \in R_l$ is a polynomial $\alpha_u(x) \in F_p[x]$ of degree less than m_l. Compute α'_u as, $\alpha'_u = \alpha_u(p_l(x)) \mod f$, for $1 \leq u \leq t$, and construct the polynomial $h'_l(y) = y^t + \alpha'_1 y^{t-1} + \ldots + \alpha'_t \in R[y]$.
  (5) (*Construct graph* G_l *implicitly*) If $l = 1$ then assign $g_l(y) = h'_l(y) \in R[y]$ and continue the loop with the next value of l. Else, construct the polynomial $h'_l(p_l(y))$ by replacing y by $p_l(y)$ in $h_l(y)$ and compute $g_l(y)$ as,
  $$g_l(y) = gcd(g_{l-1}(y), h'_l(p_l(y))) \in R[y].$$
  (6) (*Check if* G_l *is a null graph*) Let $g_l(y) = \beta_{t'} y^{t'} + \ldots + \beta_0$, where t' is the degree of $g_l(y)$ and $\beta_u \in R$ for $0 \leq u \leq t'$. If $t' = 0$ then make $g_l(y) = g_{l-1}(y)$ and continue the loop with the next value of l.

  [Steps (7) - (8): Checking for equal out degrees of the vertices of graph G_l.]

  (7) (*Check if out degrees are equal*) Else, $t' > 0$. If $\beta_{t'}$ is a zero divisor in R, construct a proper factor of f from $\beta_{t'}$ and stop.
  (8) (*Factor if out degrees are small*) Else, if $t' \leq c$ then use Evdokimov's algorithm [Evd94] on $g_l(y)$ to find a proper factor of f in $(n \log p)^{O(1)}$ time.

  [Steps (9) - (11): Checking for equal in degrees of the vertices of graph G_l.]

(9) (*Obtain the values of a nice polynomial at multiple points*) If $t' > c$, evaluate $g_l(y) \in R[y]$ at $n \cdot t'$ distinct points $y_1, \ldots, y_{nt'}$ taken from F_p. Find the characteristic polynomials of elements $g_l(y_1), \ldots, g_l(y_{nt'}) \in R$ over F_p as $c_1(x), \ldots, c_{nt'}(x) \in F_p[x]$, respectively. Collect the terms $c_i(0)$ for $1 \leq i \leq nt'$.

(10) (*Construct the nice polynomial from the values*) Construct the polynomial $r(x) = x^{nt'} + r_1 x^{nt'-1} + \ldots + r_{nt'} \in F_p[x]$ such that $r(y_i) = -c_i(0)$ for $1 \leq i \leq nt'$. Solve for $r_i \in F_p$, $1 \leq i \leq nt'$, using linear algebra.

(11) (*Check if in degrees are equal*) For $0 \leq i < t'$, if $f^i(x)$ divides $r(x)$ then compute $gcd\left(\frac{r(x)}{f^i(x)}, f(x)\right) \in F_p[x]$. If a proper factor of f is found, stop. Else, continue with the next value of l.

endfor

- If a proper factor of f is *not* found in the above *for* loop, return "Failure".

Theorem 3.2. *Algorithm 2 fails to find a proper factor f in $k \cdot (n \log p)^{O(1)}$ time if and only if f is k-cross balanced and regularity of graph G_k is greater than c.*

Proof: We show that, Algorithm 2 fails to find a proper factor of f at the l^{th} iteration of the loop iff f is l-cross balanced and regularity of G_l is greater than c. Recall the definitions of the sets $\Delta_i^{(l)}$ and $D_i^{(l)}$, $1 \leq i \leq n$, from section 1. The set $\Delta_i^{(l)}$ is defined as,

$$\Delta_i^{(l)} = \{1 \leq j \leq n : p_l(\xi_i) \neq p_l(\xi_j), \sigma((p_l(\xi_i) - p_l(\xi_j))^2) = -(p_l(\xi_i) - p_l(\xi_j))\}$$

And set $D_i^{(l)}$ is defined iteratively over l as,

$$\begin{aligned} D_i^{(1)} &= \Delta_i^{(1)} \\ \text{For } l > 1,\ D_i^{(l)} &= D_i^{(l-1)} \cap \Delta_i^{(l)} \\ \text{If } D_i^{(l)} &= \phi \text{ for all } i,\ 1 \leq i \leq n, \text{ then } D_i^{(l)} \text{ is redefined as } D_i^{(l)} = D_i^{(l-1)}. \end{aligned}$$

Graph G_l, with n vertices $v_1, \ldots, v_n$, has an edge from v_i to v_j iff $j \in D_i^{(l)}$.

Algorithm 2 fails at the first iteration ($l = 1$) if and only if f is square balanced. In this case, $D_i^{(1)} = \Delta_i^{(1)} = \Delta_i$, the polynomial $g_1(y)$ is,

$$g_1(y) = h(y) = \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(1)}} (y - \xi_j) \right] \chi_i$$

and G_1 is regular with in degree and out degree of a vertex v_i equal to $\#D_i^{(1)} = \#\Delta_i = \frac{n-1}{2}$. Thus, polynomial f is 1-cross balanced and $deg(g_1(y)) = \frac{n-1}{2}$. If Algorithm 2 fails at the l^{th} iteration, then we can assume that the polynomials $f = \tilde{f}_1, \ldots, \tilde{f}_l$ are square free and square balanced (by Lemma 3.1).

Suppose that, Algorithm 2 fails at the l^{th} iteration. Then, $\tilde{f}_l = \prod_{j=1}^{m_l} \left(x - p_l(\xi_{s_j})\right)$ is square free and square balanced, and Algorithm 1 returns the polynomial $h_l(y) \in R_l[y]$ such that,

$$h_l(y) = \sum_{j=1}^{m_l} \left[\prod_{r \in \tilde{\Delta}_j^{(l)}} (y - p_l(\xi_{s_r})) \right] \chi_j^{(l)} \tag{3.2}$$

where $\chi_j^{(l)}$'s are the primitive idempotents of the ring $R_l = \frac{F_p[x]}{(\tilde{f}_l)}$ and,

$$\tilde{\Delta}_j^{(l)} = \{1 \leq r \leq m_l : r \neq j, \sigma((p_l(\xi_{s_j}) - p_l(\xi_{s_r}))^2) = -(p_l(\xi_{s_j}) - p_l(\xi_{s_r}))\}$$

Let, $h_l(y) = y^t + \alpha_1 y^{t-1} + \ldots + \alpha_t$, where $t = \frac{m_l - 1}{2}$ and $\alpha_u \in R_l$ for $1 \leq u \leq t$. Each $\alpha_u \in R_l$ is a polynomial $\alpha_u(x) \in F_p[x]$ with degree less than m_l and if $\alpha_u = \sum_{j=1}^{m_l} a_{uj}\chi_j^{(l)}$ for $a_{uj} \in F_p$, then by Chinese Remaindering theorem (and assuming the correspondence between $\chi_j^{(l)}$ and the factor $(x - p_l(\xi_{s_j}))$ of $\tilde{f}_l$) we get,

$$\begin{aligned}
\alpha_u(x) &= q(x)(x - p_l(\xi_{s_j})) + a_{uj} \quad \text{for some polynomial } q(x) \in F_p[x] \\
\Rightarrow \alpha_u(p_l(x)) &= q(p_l(x))(p_l(x) - p_l(\xi_{s_j})) + a_{uj} \\
\Rightarrow \alpha_u(p_l(x)) &= a_{uj} \mod (x - \xi) \quad \text{for every } \xi \in \{\xi_1, \ldots, \xi_n\} \text{ such that } p_l(\xi) = p_l(\xi_{s_j})
\end{aligned}$$

Suppose that, for a given i $(1 \leq i \leq n)$, $j(i)$ $(1 \leq j(i) \leq m_l)$ is a unique index such that, $p_l(\xi_i) = p_l(\xi_{s_{j(i)}})$. Then, the polynomial $\alpha'_u(x) = \alpha_u(p_l(x)) \mod f$ has the following *direct sum (or canonical)* representation in the ring R,

$$\alpha'_u(x) = \sum_{i=1}^{n} a_{uj(i)}\chi_i$$

This implies that the polynomial $h'_l(y) = y^t + \alpha'_1 y^{t-1} + \ldots + \alpha'_t \in R[y]$ has the *canonical* representation,

$$h'_l(y) = \sum_{i=1}^{n} \left[\prod_{r \in \tilde{\Delta}_{j(i)}^{(l)}} (y - p_l(\xi_{s_r})) \right] \chi_i \tag{3.3}$$

Inductively, assume that $g_{l-1}(y)$ has the form,

$$g_{l-1}(y) = \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(l-1)}} (y - \xi_j) \right] \chi_i$$

Then,

$$\begin{aligned}
g_l(y) &= gcd\left(g_{l-1}(y), h'_l(p_l(y))\right) \\
&= \sum_{i=1}^{n} gcd\left(\prod_{j \in D_i^{(l-1)}} (y - \xi_j), \prod_{r \in \tilde{\Delta}_{j(i)}^{(l)}} (p_l(y) - p_l(\xi_{s_r})) \right) \chi_i \\
&= \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(l-1)} \cap \Delta_i^{(l)}} (y - \xi_j) \right] \chi_i \quad (\text{as } r \in \tilde{\Delta}_{j(i)}^{(l)} \Leftrightarrow s_r \in \Delta_i^{(l)})
\end{aligned}$$

Therefore,

$$\begin{aligned}
g_l(y) &= \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(l)}} (y - \xi_j) \right] \chi_i \\
&= \beta_{t'} y^{t'} + \ldots + \beta_0 \quad (\text{say})
\end{aligned}$$

where $t' = max_i \left(\#D_i^{(l)}\right)$ and $\beta_u \in R$ for $1 \leq u \leq t' \leq \frac{n-1}{2}$. The element $\beta_{t'}$ is not a zero divisor in R if and only if $\#D_1^{(l)} = \ldots = \#D_n^{(l)} = t'$. If $t' \leq c$ then a factor of f can be retrieved from $g_l(y)$ in polynomial time using already known methods ([Evd94]). The condition $\#D_i^{(l)} = t'$ for all $i, 1 \leq i \leq t'$, makes the *out* degree of every vertex in G_l equal to t'. However, this may not necessarily imply that the *in* degree of every vertex in G_l is also t'. Checking for identical *in* degrees of the vertices of G_l is handled in steps $(9) - (11)$ of the algorithm. Consider evaluating the polynomial $g_l(y)$ at a point $y_s \in F_p$.

$$g_l(y_s) = \sum_{i=1}^{n} \left[\prod_{j \in D_i^{(l)}} (y_s - \xi_j) \right] \chi_i \in R$$

The characteristic polynomial of $g_l(y_s)$ over F_p is,

$$\begin{aligned} c_s(x) &= \prod_{i=1}^{n} \left(x - \prod_{j \in D_i^{(l)}} (y_s - \xi_j) \right) \\ \Rightarrow -c_s(0) &= \prod_{j=1}^{n} (y_s - \xi_j)^{k_j} \quad \text{(since } n \text{ is odd)} \end{aligned}$$

where k_j is the *in* degree of vertex v_j in G_l. Let $r(x) = x^{nt'} + r_1 x^{nt'-1} + \ldots + r_{nt'} \in F_p[x]$ be a polynomial of degree nt', such that,

$$r(y_s) = -c_s(0) = \prod_{j=1}^{n} (y_s - \xi_j)^{k_j}$$

for nt' distinct points $\{y_s\}_{1 \leq s \leq nt'}$ taken from F_p. Since we have assumed that $p > n^2 > \frac{n(n-1)}{2} \geq nt'$, we can solve for the coefficients $r_1, \ldots, r_{nt'}$ using any nt' distinct points from F_p. Then,

$$r(x) = \prod_{j=1}^{n} (x - \xi_j)^{k_j}$$

If $k_j \neq t'$ for some j, then there is an $i = min\{k_1, \ldots, k_n\} < t'$ such that $f^i(x)$ divides $r(x)$ and $gcd\left(\frac{r(x)}{f^i(x)}, f(x)\right)$ yields a nontrivial factor of $f(x)$. This shows that the graph G_l is regular if the algorithm fails at the l^{th} step. Since $deg(g_l(y))$ equals the regularity of G_l, hence if the latter quantity is less than c then we can apply Evdokimov's algorithm [Evd94] on $g_l(y)$ and get a non trivial factor of f in polynomial time. ■

Let H_l $(1 \leq l \leq k)$ be a digraph with n vertices $v_1, \ldots, v_n$ such that there is an edge from v_i to v_j iff $j \in \Delta_i^{(l)}$. Then, graph $G_l = G_{l-1} \cap H_l$ or $G_l = G_{l-1}$ (if $G_{l-1} \cap H_l = \Phi$, where Φ is the null graph with n vertices but no edge). Here $\cap$ denotes the edge intersection of graphs defined on the same set of vertices. Algorithm 2 fails to find a proper factor of f in polynomial time if and only if there exists an $l \leq k$ such that G_l is t-regular $(t > c)$ and $G_l \cap H_j = G_l$ or Φ for all j, $l < j \leq k$. It is therefore important to choose the polynomials $p_j(\cdot)$ in such a way that very quickly we get a graph H_j with $G_l \cap H_j \neq G_l$ or Φ. We say

that a polynomial $p_l(\cdot)$ is good if either H_l is not regular or $G_l \neq G_{l-1}$ $(1 < l \leq k)$. We show that, only a few good polynomials are required.

Lemma 3.3. *Algorithm 2 (with a slight modification) requires at most* $\lceil \log_2 n \rceil$ *good auxiliary polynomials to find a proper factor of* f.

Proof: Consider the following modification of Algorithm 2. At step 5 of Algorithm 2, for $l > 1$, take $g_l(y)$ to be either $gcd(g_{l-1}(y), h_l'(p_l(y)))$ or $g_{l-1}(y)/gcd(g_{l-1}(y), h_l'(p_l(y)))$, whichever has the smaller nonzero degree. Accordingly, we modify the definition of graph G_l. Define the set $\bar{\Delta}_i^{(l)}$ $(1 \leq i \leq n)$ as,

$$\bar{\Delta}_i^{(l)} = \{1 \leq j \leq n : j \neq i, \sigma((p_l(\xi_i) - p_l(\xi_j))^2) = (p_l(\xi_i) - p_l(\xi_j))\} = \{1 \leq j \leq n : j \neq i\} - \Delta_i^{(l)}$$

and modify the definition of the sets $D_i^{(l)}$ $(1 \leq i \leq n)$ as,

$$\begin{aligned} D_i^{(1)} &= \Delta_i^{(1)} \\ \text{For } l > 1,\ D_i{}^{(l)} &= D_i{}^{(l-1)} \cap \Delta_i^{(l)} \text{ if } g_l(y) = gcd(g_{l-1}(y), h_l'(p_l(y))) \\ &= D_i{}^{(l-1)} \cap \bar{\Delta}_i^{(l)} \text{ else if } g_l(y) = g_{l-1}(y)/gcd(g_{l-1}(y), h_l'(p_l(y))) \end{aligned}$$

As before, an edge (v_i, v_j) is present in G_l iff $j \in D_i^{(l)}$. This modification ensures that, if $g_l(y) \neq g_{l-1}(y)$ has an invertible leading coefficient (i.e if $g_l(y)$ is monic) then the degree of $g_l(y)$ is at most half the degree of $g_{l-1}(y)$. Hence, for every good choice of polynomial $p_l(\cdot)$ if G_{l-1} and G_l are t_{l-1}-regular and t_l-regular, respectively, then $t_l \leq \frac{t_{l-1}}{2}$. Therefore, at most $\lceil \log_2 n \rceil$ good choices of polynomials $p_l(\cdot)$ are required by the algorithm. ∎

Theorem 1.1 follows as a corollary to Theorem 3.2 and Lemma 3.3. As already pointed out in section 1, if only $\epsilon \lceil \log_2 n \rceil$ good auxiliary polynomials are available for some ϵ, $0 < \epsilon \leq 1$, then we obtain a nontrivial factor $g(y)$ of $f'(y)$ with degree at most $\frac{n^{1-\epsilon}}{2}$. If we apply Evdokimov's algorithm on $g(y)$ instead of $f'(y)$, then the maximum dimension of the rings considered is bounded by $n^{\frac{(1-\epsilon)^2}{2} \log n + \epsilon + O(1)}$ instead of $n^{\frac{\log n}{2} + O(1)}$ (as is the case in [Evd94]).

In the following discussion we briefly analyze the performance of Algorithm 2 based on uniform random choices of the auxiliary polynomials $p_l(.)$ $(1 < l \leq k)$. The proofs are omitted.

Lemma 3.4. *If* $p = 3 \mod 4$ *and* $p \geq n^6 2^{2n}$ *then about* $\frac{(1+o(1))^n}{(\frac{\pi}{2} n)^{\frac{n}{2}}}$ *fraction of all completely splitting, square-free polynomials of degree* n *are square balanced.*

Corollary 3.5. *If* $p = 3 \mod 4$, $p > n^6 2^{2n}$ *and* $p_l(y)$ *is a uniformly randomly chosen polynomial of degree* $(n-1)$ *then the probability that* f_l *is either not square-free or is a square-free and square balanced polynomial is upper bounded by* $\frac{(1+o(1))^n}{(\frac{\pi}{2} n)^{\frac{n}{2}}}$.

It follows that, for $p = 3 \mod 4$ and $p > n^6 2^{2n}$, if the auxiliary polynomials $p_l(\cdot)$'s are uniformly randomly chosen then Algorithm 2 works in randomized polynomial time. However, the arguments used in the proof of Lemma 3.4 do not immediately apply to the case $p = 1 \mod 4$. Therefore, we resort to a more straightforward analysis, although in the process we get a slightly weaker probability bound.

Lemma 3.6. *If G_l $(1 \leq l < k)$ is regular and $p_{l+1}(y) \in F_p[y]$ is a uniformly randomly chosen polynomial of degree $(n-1)$ then $G_{l+1} \neq G_l$ with probability at least $1 - \frac{1}{2^{0.9n-2}}$.*

Thus, if polynomials $p_l(y)$, $1 < l \leq \lceil \log_2 n \rceil$, are randomly chosen, then the probability that f is not factored by Algorithm 2 within $\lceil \log_2 n \rceil$ iterations is less than $\frac{\lceil \log_2 n \rceil}{2^{0.9n-2}}$.

4. Conclusion

In this paper, we have extended the square balance test by Gao [Gao01] and showed a direction towards improving the time complexity of the best previously known deterministic factoring algorithms. Using certain auxiliary polynomials, our algorithm attempts to exploit an inherent asymmetry among the roots of the input polynomial f in order to efficiently find a proper factor. The advantage of using auxiliary polynomials is that, unlike [Evd94], it avoids the need to carry out computations in rings with large dimensions, thereby saving overall computation time to a significant extent. Motivated by the stringent symmetry requirement from the roots of f, we pose the following question:

- Is it possible to construct good auxiliary polynomials in deterministic polynomial time?

An affirmative answer to the question will immediately imply that factoring polynomials over finite fields can be done in deterministic polynomial time under ERH.

Acknowledgements

The author would like to thank Manindra Agrawal and Piyush Kurur for many insightful discussions that helped in improving the result. The suggestions from anonymous referees have significantly improved the presentation of this paper. The author is thankful to them.

References

[Ber70] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24(111):713–735, 1970.

[CH00] Qi Cheng and Ming-Deh A. Huang. Factoring polynominals over finite fields and stable colorings of tournaments. *ANTS*, pages 233–246, 2000.

[CZ81] David G. Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

[Evd94] Sergei Evdokimov. Factorization of polynominals over finite fields in subexponential time under GRH. *ANTS*, pages 209–219, 1994.

[Gao01] Shuhong Gao. On the deterministic complexity of factoring polynomials. *Journal of Symbolic Computation*, 31(1–2):19–36, 2001.

[KS95] Erich Kaltofen and Victor Shoup. Subquadratic-time factoring of polynomials over finite fields. *STOC*, pages 398–406, 1995.

[LN94] R. Lidl and H. Niederreiter. Introduction to finite fields and their applications, revised edition. *Cambridge University Press*, 1994.

[vzGS92] Joachim von zur Gathen and Victor Shoup. Computing frobenius maps and factoring polynomials. *Computational Complexity*, 2:187–224, 1992.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 621-632
www.stacs-conf.org

ON THE DECOMPOSITION OF k-VALUED RATIONAL RELATIONS

JACQUES SAKAROVITCH [1] AND RODRIGO DE SOUZA [2]

[1] LTCI, ENST/CNRS, Paris (France)
E-mail address: sakarovitch@enst.fr

[2] ENST, 46, rue Barrault, 75634 Paris Cedex 13 (France)
E-mail address: rsouza@enst.fr

Abstract. We give a new, and hopefully more easily understandable, structural proof of the decomposition of a k-valued transducer into k unambiguous functional ones, a result established by A. Weber in 1996. Our construction is based on a lexicographic ordering of computations of automata and on two coverings that can be build by means of this ordering. The complexity of the construction, measured as the number of states of the transducers involved in the decomposition, improves the original one by one exponential. Moreover, this method allows further generalisation that solves the problem of decomposition of rational relations with bounded length-degree, which was left open in Weber's paper.

1. Introduction

This communication is part of a complete reworking[1] and rewriting of the theory of k-valued rational relations and transducers which puts it in line with the theory of rational functions (1-valued rational relations) and functional transducers and makes it appear as a natural generalisation of the latter not only at the level of the results — as we recall in the next paragraph — but also *at the level of proofs*.

It is decidable whether a transducer is functional (originally due to Schützenberger [13]); as a consequence, the equivalence of functional transducers is decidable, and, above all, every functional transducer is equivalent to an unambiguous one [5]. These results generalise in a remarkable way to bounded valued rational relations and transducers. It is decidable whether the image of every word by a given transducer is bounded (Weber [14]), it is decidable whether it is bounded by a given integer k (Gurari and Ibarra [6]), every k-valued transducer is equivalent to the sum of k functional (and thus unambiguous) ones (Weber [15]) and the equivalence of k-valued transducers is decidable (Culik and Karhumäki [4]).

It is noteworthy that all the results just quoted for functional transducers are now (if not in the original papers) established by means of constructions conducted on the transducers themselves [2,9,11] whereas the corresponding results on k-valued transducers

1998 ACM Subject Classification: F.1.1, F.4.3.

Key words and phrases: rational relation, k-valued transducer, unambiguous transducer, covering of automata.

[1]A financial support of CAPES Foundation (Brazilian government) for doctoral studies is gratefully acknowledged by the second author (second in the alphabetical order, as in use in the British and French encyclopedias — not in the Lusitanian ones).

come, in some sense, "from outside" and, what is worse, from a different world for each of them. Gurari and Ibarra's proof for the decidability of the k-valuedness relies on a reduction to the emptiness problem for a class of counter automata, Culik and Karhumäki's one for the decidability of the equivalence appears in the context of the solution of Ehrenfeucht's conjecture on HDTOL languages, and Weber's proof of the decomposition — which we shall discuss more in detail below — is highly combinatorial and still somewhat detached from the transducers.

Our approach for those results are based on constructions which depend directly on the structure of the automata. They give back the subject a full coherence and yield systematically better complexity bounds. This will be illustrated in this paper with a new proof of the decomposition theorem which we restate below as Theorem 1.1. In [12] we give a new proof for the decidability of the k-valuedness.

Theorem 1.1 (Weber [15])**.** *Every k-valued transducer $\mathcal{T}$ can be effectively decomposed into a sum of k (unambiguous) functional transducers.*[2]

Our proof for Theorem 1.1 differs from the original one by three aspects. First, Weber's proof is generally considered as very difficulty to follow, whereas ours is hopefully simpler. Second, Weber's construction results in k transducers whose number of states is a double exponential on the number of states of $\mathcal{T}$, whereas we obtain a decomposition of single exponential size. Third and finally, our method allows to solve the problem, posed by Weber, of the decomposition of bounded length-degree rational relations with a more general statement (in Weber's question, θ is the length morphism):

Theorem 1.2. *Let $\tau : A^* \to B^*$ be a finite image rational relation and $\theta : B^* \to C^*$ a morphism such that the composition $\tau\theta$ is k-valued.*[3] *Every transducer $\mathcal{S}$ realising τ can be effectively decomposed into k transducers whose compositions with θ are functions.*

Our proof makes use twice of the notion of covering of automata. A covering of an automaton[4] $\mathcal{A}$ is an *expansion* of $\mathcal{A}$: a new automaton $\mathcal{B}$ whose states and transitions map to those of $\mathcal{A}$, preserving adjacency and labels of transitions. Moreover, the outgoing transitions of every state of $\mathcal{B}$ map one-to-one to those of the projection, which implies a bijection between the successful computations of $\mathcal{A}$ and $\mathcal{B}$. Typically, $\mathcal{B}$ is larger than $\mathcal{A}$, for several states can have the same image. This allows to choose certain subsets of the computations of $\mathcal{A}$ by erasing parts of $\mathcal{B}$.

The two coverings we are going to define are based on a lexicographic ordering on the computations. This method can be seen as a conceptual generalisation of the one used by H. Johnson in order to build a lexicographic selection of deterministic rational relations [7, 8].

The first construction, explained in Section 3.3, is what we call the *lag separation covering* $\mathcal{U}_N$ of a (real time) transducer $\mathcal{T}$. It is parameterised by an integer N, and roughly speaking allows to distinguish between computations with same input and same output and

[2]By "decomposed" we mean that the relation realised by $\mathcal{T}$ and the union of the relations realised by the k transducers are the same.

[3]We write functions and relations using a postfix notation: $x\tau$ is the image of x by the relation τ and thus the composition of relations is written by left-to-right concatenation. Let us recall that the rational relations are closed under composition [5].

[4]As we shall define in Section 2, transducers are automata of a certain kind.

whose *lag*[5] is bounded by N. If $\mathcal{T}$ is k-valued, we show that for a certain N, $\mathcal{U}_N$ contains a subtransducer $\mathcal{V}_N$ which is equivalent to $\mathcal{T}$ and input-k-ambiguous[6] (Proposition 4.2).

The second construction (Section 3.2) is what we call the *multi-skimming covering* of an $\mathbb{N}$-automaton. It proves the following *multi-skimming theorem* for $\mathbb{N}$-rational series:

Theorem 1.3. *Let $\mathcal{A}$ be a finite $\mathbb{N}$-automaton with n states realising the series s. There exists an infinite $\mathbb{N}$-covering $\mathcal{B}$ of $\mathcal{A}$ such that for every integer $k > 0$, there exists a finite $\mathbb{N}$-quotient $\mathcal{B}_k$ of $\mathcal{B}$ which satisfies: $\mathcal{B}_k$ is an $\mathbb{N}$-covering of $\mathcal{A}$ with at most $n\,(k+1)^n$ states; for every i, $0 \leq i < k$, there exists an unambiguous subautomaton $\mathcal{B}_k^{(i)}$ of $\mathcal{B}_k$ which recognises the support of $s - i$; there exists a subautomaton $\mathcal{D}_k$ of $\mathcal{B}_k$ whose behaviour is $s - k$.*

Here $s - k$ is the series obtained from s by subtracting k to every coefficient larger than k and assigning 0 to the others. In particular, Theorem 1.3 says that, if $\mathcal{A}$ is a k-ambiguous automaton, then there exists a finite covering $\mathcal{B}_k$ of $\mathcal{A}$ and unambiguous subautomata $\mathcal{B}_k^{(0)}, \ldots, \mathcal{B}_k^{(k-1)}$ of $\mathcal{B}_k$ such that the successful computations of the union $\bigcup_i \mathcal{B}_k^{(i)}$ are in bijection with those of $\mathcal{A}$. Of course, it is not new that $s - k$ is a $\mathbb{N}$-rational series when s is. This is an old result by Schützenberger which can be proved by iterated applications of Eilenberg's Cross-Section Theorem [5], or of the construction given in [11]. But all these methods yield an automaton whose size is a tower of exponentials of height k. Theorem 1.3 thus answers a problem left open in [11] with a solution which is better than the one that was conjectured there.

These coverings together give in two steps a decomposition of a k-valued transducer $\mathcal{T}$. First, the lag separation covering of $\mathcal{T}$ yields a transducer $\mathcal{V}_N$ equivalent to $\mathcal{T}$ and whose underlying input automaton, say $\mathcal{A}$, is k-ambiguous. Next, the multi-skimming covering applied to $\mathcal{A}$ yields, as stated in the discussion after Theorem 1.3, k unambiguous automata $\mathcal{B}_k^{(i)}$; the successful computations of the union of the $\mathcal{B}_k^{(i)}$ are in bijection with those of $\mathcal{A}$ and, as the transitions of every $\mathcal{B}_k^{(i)}$ map on those of $\mathcal{A}$, one can "lift" on them the output of the corresponding transitions of $\mathcal{V}_N$: one thus obtain k unambiguous functional transducers $\mathcal{Z}^{(0)}, \ldots, \mathcal{Z}^{(k-1)}$ decomposing $\mathcal{T}$ (see Figure 1).

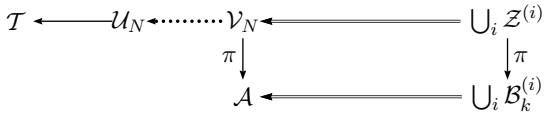


Figure 1: Decomposition of a k-valued transducer $\mathcal{T}$. The simple edge stands for a covering; the dotted one represents an input-k-ambiguous subautomaton; the double ones are immersions; π is a projection on the underlying input automaton.

Our proof goes so to speak in the opposite way that Weber's one: our first step is to build an input-k-ambiguous transducer from which the decomposition is extracted, whereas the existence of such a transducer is viewed in [15] as a consequence of the decomposition. Moreover, although both proofs have a very general idea in common — a classification of computations from which at most k successful ones can be distinguished, for every input word — the way we do this is completely different. Indeed, Weber's decomposition is

[5]To be defined in the body of the paper.

[6]When it comes to ambiguity in transducers, we distinguish between *input-ambiguity* (called ambiguity in most of the references) and ambiguity of the transducer (which allows to define ambiguity for relations).

extracted from the strongly connected components of a graph built on a preliminary decomposition of $\mathcal{T}$ into exponentially many functional transducers. We perform a selection among the computations of $\mathcal{T}$ according to a lexicographic ordering on the transitions.

A rough estimation of the complexity (number of states, as a function on the number of states of $\mathcal{T}$) of this two-step procedure gives a double exponential: one for the lag separation covering $\mathcal{U}_N$ and other for the multi-skimming covering. However, a major feature of this construction is that every computation in the newly built automata corresponds to a computation in the original transducer — this is basically what we mean by structural proof — which allows to track down the usefulness of every newly created state. Then, a careful analysis shows that restricting the constructions to the *trim* parts of the automata the number of obtained states is bounded by $2^{\mathcal{O}(hLk^4n^{k+4})}$ states, where n is the number of states of $\mathcal{T}$, h is the size of the output alphabet and L is the maximal length of the outputs of the transitions (Section 4.2). This is to be compared with the size of Weber's decomposition described in [15], 2^{2^P}, where $P = p(n + L + h + k)$ is a polynomial whose degree and coefficients do not seem to be easily derived from the arguments developed there.

The proof of Theorem 1.2 starts with the construction of k unambiguous transducers decomposing the k-valued relation $\tau\theta$. Next, we show that these transducers induce a decomposition of the set of successful computations of $\mathcal{S}$. This gives a new set of k finite transducers, not necessarily unambiguous, which decompose $\mathcal{S}$ (Section 4.3).

Finally, let us note that — as explained in [15] — the improvement in the size of the decomposition from double to single exponential yields an improvement of the same order for the complexity of the decision of the equivalence of k-valued transducers.

2. Preliminaries

We basically follow the definitions and notation in [1, 5, 10].

The semiring of the nonnegative integers is denoted by $\mathbb{N}$, the set of words over a finite alphabet A (the free monoid over A) by A^* and the empty word by 1_{A^*}. The length of $u \in A^*$ is denoted by $|u|$. The powerset of a set X is denoted by $\mathfrak{P}(X)$.

An *automaton* over a monoid M is a labelled directed graph $\mathcal{A} = (Q, M, E, I, T)$ defined by the set Q of vertices, called *states* and E of edges, called *transitions*, together with two subsets I and T of Q, the initial and final states respectively. Every transition e in E is associated with a triple (p, m, q) of $Q \times M \times Q$, specifying its origin, label and end. Note that we shall explicitly consider cases where *distinct transitions* have the *same* origin, label and end, even though we take the liberty to write $e : p \xrightarrow{m} q \in E$ meaning a transition e associated with (p, m, q). The automaton $\mathcal{A}$ is *finite* if Q and E are finite.

A *computation* in $\mathcal{A}$ is a sequence of transitions $c : p_0 \xrightarrow{m_1} p_1 \xrightarrow{m_2} \dots \xrightarrow{m_l} p_l$, also denoted as $p_0 \xrightarrow[\mathcal{A}]{m_1 \dots m_l} p_l$. Its label is $m_1 \dots m_l \in M$ and its length l. It is *successful* if $p_0 \in I$ and $p_l \in T$. The *behaviour* of $\mathcal{A}$ is the set $|\mathcal{A}| \subseteq M$ of labels of successful computations. These sets are the family $\operatorname{Rat} M$ of the *rational subsets* of M.

A state of $\mathcal{A}$ is *accessible* if it can be reached by a computation starting at some state of I, and *co-accessible* if some state of T can be reached from it. The state is *useful* if is both accessible and co-accessible, and we say that $\mathcal{A}$ is *trim* if every state is useful.

If M is a free monoid A^* and the labels of transitions are letters, then $\mathcal{A}$ is a classical automaton over A; we write in this case $\mathcal{A} = (Q, A, E, I, T)$. If M is a product $A^* \times B^*$, then every transition is labelled by a pair denoted as $u|x$ and consisting of an input word

$u \in A^*$ and an output one $x \in B^*$; and $\mathcal{A}$ is a *transducer* realising a *rational relation* from A^* to B^*. The image of a word $u \in A^*$ by a transducer is the set of outputs of successful computations whose input is u. The transducer is called *k-valued*, for $k \in \mathbb{N}$, if the cardinality of the image of every input word is at most k.

By using classical constructions on automata, every transducer can be transformed into a *real-time* one: a transducer whose labels are of form $a|K$, where a is a *letter*, $K \in \operatorname{Rat} B^*$ and I and T are functions from Q to $\operatorname{Rat} B^*$ [5, 10]. For finite image relations we may suppose that the transitions read a letter and output a single word, and the image of every final state is 1_{B^*}. In this case, the transducer is denoted rather as $\mathcal{T} = (Q, A, B^*, E, I, T)$.

The *underlying input automaton* $\mathcal{A}$ of a real-time transducer $\mathcal{T}$ is the (classical) automaton obtained by forgetting the output of the transitions and replacing the functions I and T by their domains. The behaviour of $\mathcal{A}$ is the domain of the relation realised by $\mathcal{T}$.

Figure 2: A 2-valued real-time transducer $\mathcal{T}$ over $\{a\}^* \times \{b\}^*$ and its (infinitely ambiguous) underlying input automaton. The behaviour of $\mathcal{T}$ is the relation defined by $(1_{A^*})|\mathcal{T}| = 1_{B^*}$ and $(a^n)|\mathcal{T}| = \{b^n, b^{n+1}\}$ for $n > 0$.

An $\mathbb{N}$-*automaton* is an automaton labelled by letters with multiplicities in $\mathbb{N}$ attached to the transitions and to initial and final states. It realises an $\mathbb{N}$-*rational series*: a function $s : A^* \to \mathbb{N}$ which assigns to $u \in A^*$ a multiplicity given by summing the multiplicites (product of the multiplicities of transitions) of the successful computations labelled by u.

Every $\mathbb{N}$-automaton or real-time transducer can be described by a *matrix representation* (λ, μ, ν), where $\lambda \in S^Q$ ($\nu \in S^Q$) is a row (column) vector for the multiplicities of the initial (final) states, $\mu : A^* \to S^{Q \times Q}$ is a morphism, $S = \mathbb{N}$ for $\mathbb{N}$-automata and $S = \operatorname{Rat} B^*$ for transducers. The behaviour can be expressed by the function which maps every $u \in A^*$ to $\lambda \cdot u\mu \cdot \nu$. This leads to call *dimension* the set of states of an automaton.

It will be useful to consider $\mathbb{N}$-automata whose transitions are *characteristic*, that is, with multiplicity 1. Every $\mathbb{N}$-automaton can be transformed into such a one by splitting every transition with multiplicity $l > 0$ into a set of l characteristic ones (Figure 3).

Figure 3: An $\mathbb{N}$-automaton $\mathcal{C}_1$ over $\{a, b\}$, on the right-hand side with characteristic transitions obtained by splitting multiplicities. If $u \in \{a, b\}^*$ is viewed as the writing in the binary system of an integer $\overline{u}$ by interpreting a as 0 and b as 1, then $u\|\mathcal{C}_1| = \overline{u}$.

A *morphism* from $\mathcal{B} = (R, M, F, J, U)$ to $\mathcal{A} = (Q, M, E, I, T)$, denoted by $\varphi : \mathcal{B} \to \mathcal{A}$, is a pair of mappings $R \to Q$ and $F \to E$, both denoted by φ, such that $J\varphi \subseteq I$, $U\varphi \subseteq T$ and for every $e \in F$, if e is associated with (p, m, q), then $e\varphi$ is associated with $(p\varphi, m, q\varphi)$. We say that φ is a *covering* if φ induces a bijection between the outgoing transitions of p and $p\varphi$, I is in bijection with J and $T\varphi^{-1} = U$. An *immersion* is by definition a subautomaton of a covering. These conditions imply that every successful computation of $\mathcal{B}$ maps to a

successful computation of $\mathcal{A}$, and thus $|\mathcal{B}| \subseteq |\mathcal{A}|$. In the case of coverings, there is indeed a bijection between the successful computations and thus $|\mathcal{B}| = |\mathcal{A}|$ [9].

A covering of the split form of an $\mathbb{N}$-automaton $\mathcal{A}$ is the split form of an $\mathbb{N}$-*covering* of $\mathcal{A}$, see [3, 10, 11] for the definition of the latter. The $\mathbb{N}$-series realised by an $\mathbb{N}$-automata and any of its $\mathbb{N}$-coverings are the same.

3. Lexicographic coverings

The idea of the two coverings we are going to define is to order lexicographically computations of automata, inasmuch as it can be done with words on some alphabet. Here, the alphabet is the set of transitions, and computations are seen as words on it.

3.1. The lexicographic ordering of computations

Let $\mathcal{A} = (Q, A, E, I, T)$ be a classical automaton. Fix a (partial) ordering $\prec$ on E such that transitions are comparable iff they have the same label and origin. This ordering is extended on E^* and thus on the computations of $\mathcal{A}$ in such a way that it can be called a *lexicographic ordering of the computations*: $c = e_1 e_2 \dots e_l e_{l+1} \dots e_n$ and $d = e'_1 e'_2 \dots e'_l e'_{l+1} \dots e'_m$ ($e_i, e'_j \in E$ for $1 \leq i \leq n$ and $1 \leq j \leq m$) are such that $c \prec d$ iff c and d have the same label (thus $m = n$) and there exists l such that $e_i = e'_i$ for $1 \leq i \leq l-1$ and $e_l \prec e'_l$.

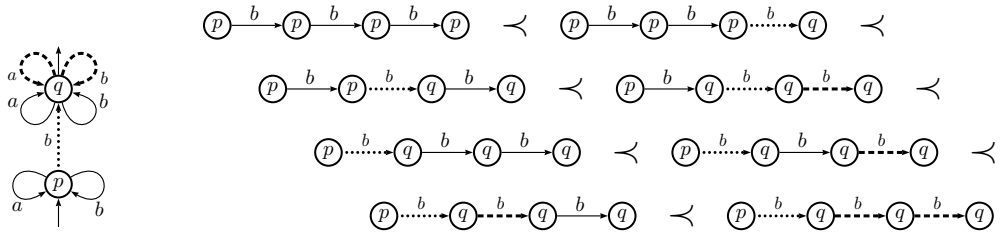


Figure 4: A lexicographic ordering between the computations of $\mathcal{C}_1$ labelled by *bbb* and starting at p. Solid transitions are smaller than the dotted and dashed ones.

The definitions for other kinds of automata are similar but, in order to give them the wanted meaning, a little bit more delicate: for $\mathbb{N}$-automata, the ordering is put on the split form, and for real-time transducers, on the underlying input automaton.[7]

3.2. The multi-skimming covering of an $\mathbb{N}$-automaton

The aim of the multi-skimming covering of an $\mathbb{N}$-automaton $\mathcal{A} = (Q, A^*, E, i, T)$ is to count, for every successful computation, the number of the smaller ones according to $\prec$.

Let $\xi : E \to \mathbb{N}^Q$ be the function from transitions to $\mathbb{N}$-vectors indexed by Q defined by $(e\xi)_r = \operatorname{card}\Big(\{f \in E \mid f : p \xrightarrow{a} r \text{ and } f \prec e\}\Big)$, for $e : p \xrightarrow{a} q \in E$ and $r \in Q$.

Definition 3.1. The multi-skimming covering of $\mathcal{A}$ is the (infinite) $\mathbb{N}$-automaton $\mathcal{B}$ of dimension $Q \times \mathbb{N}^Q$ defined as follows:

[7] In order to ease the explanation, we shall describe the constructions for automata with a single initial state. Computations starting at distinct initial states become ordered by extending $\prec$ to new transitions $i \xrightarrow{1} p$ starting at a "hidden" initial state i, for every $p \in I$. Initial multiplicities can be treated similarly.

- the initial state is $(i, \vec{0})$ (where $\vec{0}$ is the zero vector);
- the final states are $T \times \mathbb{N}^Q$;
- for every $(p, \mathsf{v}) \in Q \times \mathbb{N}^Q$ and every $e : p \xrightarrow{a} q \in E$, $(p, \mathsf{v}) \xrightarrow{a} (q, \mathsf{v} \cdot a\mu + e\xi)$ is a transition of $\mathcal{B}$ (where μ is the morphism of the matrix representation of $\mathcal{A}$). □

It follows from this definition that for every state (p, v) of $\mathcal{B}$, the outgoing transitions of (p, v) are in bijection with those of p. Thus, the projection φ of $\mathcal{B}$ on the first component is an $\mathbb{N}$-covering of $\mathcal{A}$. The property below follows by induction on the length of computations[8]:

Property 3.2. Let $C : (i, \vec{0}) \xrightarrow[\mathcal{B}]{u} (p, \mathsf{v})$ be a computation. For every $q \in Q$, v_q is the number of computations $d : i \xrightarrow[\mathcal{A}]{u} q$ such that $d \prec C\varphi$ (where $C\varphi$ is the projection of C on $\mathcal{A}$). ∎

We define as above the (finite) automaton $\mathcal{B}_k$ satisfying Theorem 1.3; the difference is that it counts *until* $k-1$. Let $\mathbb{N}_k = \{0, \ldots, k-1, \omega\}$ be the quotient semiring of $\mathbb{N}$ given by the relation $k = k+1$ (ω is the class of k and plays the role of an infinity). The dimension of $\mathcal{B}_k$ is $Q \times \mathbb{N}_k^Q$; transitions and initial and final states are defined as in Definition 3.1, but the matrix operations $\mathsf{v} \cdot a\mu + e\xi$ are made in $\mathbb{N}_k$. The morphism $\mathbb{N} \to \mathbb{N}_k$ induces an $\mathbb{N}$-quotient $\mathcal{B} \to \mathcal{B}_k$, and as noted, $\mathcal{B}_k$ is an $\mathbb{N}$-covering of $\mathcal{A}$. Figure 5 shows an example.

By induction on the length of computations, we have:

Property 3.3. Let $C : (i, \vec{0}) \xrightarrow[\mathcal{B}_k]{u} (p, \mathsf{v})$ be a computation. For every $q \in Q$, v_q is the number of computations $d : i \xrightarrow[\mathcal{A}]{u} q$ such that $d \prec C\varphi$, if this number is smaller than k, or it is ω otherwise. ∎

Proof of Theorem 1.3. In view of Property 3.3, we can obtain the subautomata $\mathcal{B}_k^{(i)}$ of $\mathcal{B}_k$ by erasing the condition of being final of some final states of $\mathcal{B}_k$: each $\mathcal{B}_k^{(i)}$ is defined by choosing as final only the states $(p, \mathsf{v}) \in T \times \mathbb{N}_k^Q$ such that $\sum_{q \in T} \mathsf{v}_q = i$; $\mathcal{D}_k$ is the subautomaton of $\mathcal{B}_k$ defining as final the states $(p, \mathsf{v}) \in T \times \mathbb{N}_k^Q$ such that $\sum_{q \in T} \mathsf{v}_q = \omega$. ∎

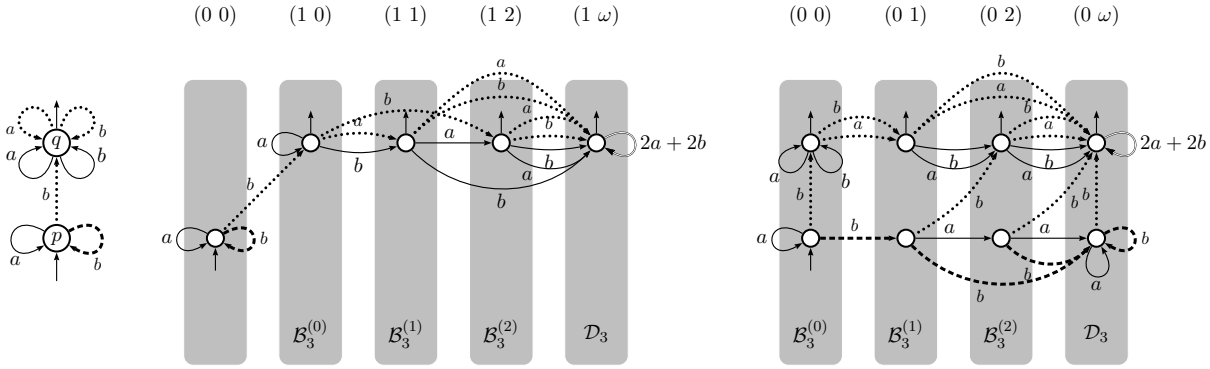
3.3. The lag separation covering of a real-time transducer

Let $\mathcal{T} = (Q, A, B^*, E, i, T)$ be a real-time transducer. We aim with the lag separation covering of $\mathcal{T}$ at a selection between computations of this transducer with *same input and same output* (stated in Property 3.9). This will be useful in Section 4 to construct a input-k-ambiguous transducer from a k-valued one.

It is not possible in general to build a finite expansion which allows to select exactly one computation for each pair of words in the relation realised by the transducer, for this would lead to an unambiguous transducer and there exist rational relations which are inherently ambiguous. The idea is to fix a parameter N and compare only computations such that *the differences of lengths of outputs along them (their "lag") are bounded by* N.

At first, let us recall the *Lead or Delay action*, defined in [2] to describe differences of words. We restate it in a slightly different form, based on the *free group* $F(B)$ generated by B: the quotient of $(B \cup \overline{B})^*$ by the relations $x\overline{x} = \overline{x}x = 1_{B^*}$ ($x \in B$), where $\overline{B}$ a disjoint copy of B. The inverse of $u \in B^*$, denoted by $\overline{u}$, is the mirror image of u with barred letters.

[8] Computations of coverings will be represented with capital letters.



(a) With the dashed transition starting at p being smaller than the dotted one...

(b) ... and with the other ordering on the outgoing transitions of p.

Figure 5: The multi-skimming covering at layer $k = 3$ for $\mathcal{C}_1$ with two different orderings of the transitions starting at p. States of the covering are pairs (r, v), where r is a state of the automaton (horizontal projection) and v is an $\mathbb{N}_3$-vector indexed by $\{p, q\}$ in that order (vertical projection). Solid transitions leaving q are smaller than the dotted ones in both coverings. The double transitions stand for four transitions. The automata $\mathcal{B}_3^{(i)}$ recognising the support of $s-i$ and $\mathcal{D}_3$ recognising $s-3$ are given by keeping as final exactly one final state at the indicated columns.

We denote by $\mathbf{1}$ the empty word of $F(B)$ (which is the class of the empty word of B). Let $\Delta = B^* \cup \overline{B}^* \cup \{\mathbf{0}\}$, where $\mathbf{0}$ is a new element, a zero, not in $F(B)$, and $\rho : F(B) \cup \{\mathbf{0}\} \to \Delta$ be the function $w\rho = w$, if $w \in \Delta$, and $w\rho = \mathbf{0}$ otherwise.

Definition 3.4. The Lead or Delay Action of $B^* \times B^*$ on Δ is defined by $w \cdot (x, y) = (\overline{x}\, w\, y)\rho$, $w \in \Delta$, $(x, y) \in B^* \times B^*$ (the product is taken with the rules $\mathbf{0}\, x = x\, \mathbf{0} = \mathbf{0}$). □

Intuitively, $\mathbf{1} \cdot (x, y)$ represents the "difference" of the words x and y, being a positive word if x is a prefix of y (the *lead* of y with respect to x), a negative word if y is a prefix of x (the *delay* of y with respect to x), and $\mathbf{0}$ if x and y are not prefixes of a common word.

Definition 3.5. Let $c : p \xrightarrow{u|x} q$ and $d : p' \xrightarrow{u|y} q'$ be two computations of $\mathcal{T}$ with the same input u. As $\mathcal{T}$ is a real-time transducer, c and d have the same length. We define their *Lead or Delay*, denoted by $\mathrm{LD}(c, d)$, as the element $\mathbf{1} \cdot (x, y)$ of Δ, and if $\mathrm{LD}(c, d) \neq \mathbf{0}$, their *lag* as the integer $\langle c, d \rangle = \mathsf{max}\{|\mathrm{LD}(c', d')| \mid c', d' \text{ prefixes of } c, d \text{ with the same length}\}$. □

Similarly to the multi-skimming covering, the states of the lag separation covering of $\mathcal{T}$ carry vectors indexed by Q. But in this case the "stored information" is the Lead or Delay between any computation and those which are smaller. Let $\xi : E \to \mathfrak{P}(\Delta)^Q$ be the function given by $(e\xi)_r = \{(\overline{x}y)\rho \mid f : p \xrightarrow{a|y} r \in E,\ f \prec e\}$, for $e : p \xrightarrow{a|x} q \in E$ and $r \in Q$.

Definition 3.6. The lag separation covering of $\mathcal{T}$ is the (infinite) real-time transducer $\mathcal{U} = (R, A, B^*, F, j, U)$ defined by

- $R = Q \times \mathfrak{P}(\Delta)^Q$;
- $j = (i, \vec{0})$ (where $\vec{0}$ is the vector whose entries are all equal to $\emptyset$);
- $U = T \times R$;

- for every $(p,\mathsf{v}) \in R$ and every $e : p \xrightarrow{a|x} q \in E$, $(p,\mathsf{v}) \xrightarrow{a|x} \big(q, (\overline{x}\cdot\mathsf{v}\cdot a\mu + e\xi)\rho\big)$ is a transition in F (where μ is the morphism of the matrix representation of $\mathcal{T}$, $\overline{x}\cdot\mathsf{v}$ is the vector obtained by multiplying on the left every entry of v by $\overline{x}$, and ρ is extended componentwise to vectors in $\mathfrak{P}(\Delta)^Q$). □

As before, for every state (p,v) of $\mathcal{U}$, there is a bijection between the outgoing transitions of (p,v) and those of p: the projection φ of $\mathcal{U}$ on the first component is a covering on $\mathcal{T}$. By induction on the length of computations, we have:

Property 3.7. Let $C : (i,\vec{0}) \xrightarrow{u|x} (p,\mathsf{v})$ be a computation of $\mathcal{U}$. For every state q of $\mathcal{T}$, v_q is the set of Lead or Delay of $C\varphi$ (the projection of C on $\mathcal{T}$) and any computation of $\mathcal{T}$ smaller than $C\varphi$ and which ends in q: $\mathsf{v}_q = \{\mathrm{LD}(C\varphi, d) \mid d : i \xrightarrow[\mathcal{T}]{u|y} q,\ d \prec C\varphi\}$. ■

In order to build the announced selection of computations of $\mathcal{T}$, we define a "bounded" lag separation covering where only the computations with lag bounded by N are compared, so that only words in $\Delta_N = B^{\leq N} \cup \overline{B}^{\leq N}$ are "stored" in the entries of the vectors v. Let $\rho_N : F(B) \to \Delta_N \cup \{\mathbf{0}\}$ be the function defined by $w\rho_N = w$, if $w \in \Delta_N$, and $w\rho_N = \mathbf{0}$ otherwise. The element $\mathbf{0}$ is intentionally omitted from Δ_N in order to simplify the writing of Property 3.8, and in the extension of ρ_N to $\mathfrak{P}(F(B))^Q$ the image of a word not in Δ_N will be seen as the empty set so that for $\mathsf{v} \in \mathfrak{P}(F(B))^Q$, $\mathsf{v}\rho_N$ is a vector in $\mathfrak{P}(\Delta_N)^Q$ (which does not contain $\mathbf{0}$ in any of its entries). We define $\mathcal{U}_N$ as the (finite) transducer constructed as in Definition 3.6, but with states and transitions given by:

$$R = Q\times\mathfrak{P}(\Delta_N)^Q, \qquad \forall\,(p,\mathsf{v}) \in R,\ \forall\, e : p \xrightarrow{a|x} q \in E \quad (p,\mathsf{v}) \xrightarrow{a|x} \big(q, (\overline{x}\cdot\mathsf{v}\cdot a\mu + e\xi)\rho_N\big) \in F.$$

Due to the fact that ρ_N is not a morphism, it is not true in general that $\mathcal{U}$ is a covering of $\mathcal{U}_N$; but $\mathcal{U}_N$ is another covering of $\mathcal{T}$. By induction we have (see Figure 6(a)):

Property 3.8. Let $C : (i,\vec{0}) \xrightarrow{u|x} (p,\mathsf{v})$ be a computation of $\mathcal{U}_N$. For every state q of $\mathcal{T}$, $\mathsf{v}_q = \{\mathrm{LD}(C\varphi, d) \mid d : i \xrightarrow[\mathcal{T}]{u|y} q,\ \ x, y \text{ prefixes of a common word},\ d \prec C\varphi,\ \langle C\varphi, d\rangle \leq N\}$. ■

The wanted selection is a consequence of Property 3.8 and can be stated as follows:

Property 3.9. Let $\mathcal{V}_N$ be the subtransducer of $\mathcal{U}_N$ obtained by removing the property of being final of every state $(p,\mathsf{v}) \in T \times R$ such that $\mathbf{1} \in \mathsf{v}_t$ for some $t \in T$. A computation C of $\mathcal{V}_N$ is successful if, and only if, $C\varphi$ is successful in $\mathcal{T}$ and for every successful computation d of $\mathcal{T}$ smaller than $C\varphi$ with (same input and) same output, $\langle C\varphi, d\rangle > N$. ■

The transducers $\mathcal{T}$ and $\mathcal{V}_N$ are equivalent: if (u,x) is in the behaviour of $\mathcal{T}$, the smallest successful computation of $\mathcal{T}$ labelled by (u,x) is the projection of a successful one in $\mathcal{V}_N$.

The following remark on the trim part of $\mathcal{V}_N$ will be useful for the evaluation of the size of the decomposition (Section 4.2).

Property 3.10. Let $\mathcal{T}$ be a trim and k-valued transducer with n states, and whose output alphabet has h letters. The number of useful states of $\mathcal{V}_N$ is bounded by 2^{2hNk^2n}.

Proof. We write $\mathfrak{P}_{(l)}(X)$ for the set of the subsets with at most l elements of a set X. Clearly, $\mathrm{card}\,\big(\mathfrak{P}_{(l)}(X)\big) \leq \mathrm{card}\,(X)^{l^2}$. The hypothesis that $\mathcal{T}$ is trim and k-valued together with Property 3.8 imply that the vectors in the useful states of $\mathcal{V}_N$ have in every coordinate

at most k words, thus these states belong to $Q \times \mathfrak{P}_{(k)}(\Delta_N)^Q$. The cardinality of this set is at most $n \cdot \left(\operatorname{card}(\Delta_N)^{k^2}\right)^n \leq n \cdot \left((2h)^{Nk^2}\right)^n$. This is clearly bounded by 2^{2hNk^2n}. ∎

4. Decomposing a k-valued rational relation

As said in the introduction, we first prove a result for k-valued transducers:

Theorem 4.1. *Any k-valued transducer is equivalent to an input-k-ambiguous one.*

This will be established by the lag separation covering: for some adequate N, $\mathcal{V}_N$ is input-k-ambiguous (Proposition 4.2). Next, Theorem 1.1 is proved by applying the multi-skimming covering on the underlying input automaton of $\mathcal{V}_N$ (Section 4.2).

4.1. From a k-valued transducer to an input-k-ambiguous one

Proposition 4.2. *Let $\mathcal{T}$ be a real-time transducer with n states and lengths of outputs of transitions bounded by L. If $\mathcal{T}$ is k-valued, then for $N \geq L\,n^{k+1}$ $\mathcal{V}_N$ is input-k-ambiguous.*

The crux of the proof is a combinatorial property stated in Theorem 2.2 of [15], and restated here as Lemma 4.3. In this lemma, $\mathcal{T}^{k+1}$ is the cartesian product of $\mathcal{T}$ by itself $k+1$ times, a natural generalisation of the squaring of $\mathcal{T}$ defined in [2] to establish the decidability of the functionality of transducers. In $\mathcal{T}^2$, every computation corresponds to a pair of computations of $\mathcal{T}$ *with the same input*; in $\mathcal{T}^{k+1}$, every computation corresponds then to a $(k+1)$ tuple of computations of $\mathcal{T}$ *with the same input* (this construction is heavily used in [12] to give a new proof of the decidability of k-valuedness).

Lemma 4.3 (Weber [15])**.** *If $\mathcal{T}$ is k-valued, then for every successful computation $\mathbf{c}$ of $\mathcal{T}^{k+1}$ there exists a pair i, j of coordinates such that the projections $\mathbf{c}_i$ and $\mathbf{c}_j$ satisfy $\mathrm{LD}(\mathbf{c}_i, \mathbf{c}_j) = 1$ (that is, $\mathbf{c}_i$ and $\mathbf{c}_j$ have the same output) and $\langle \mathbf{c}_i, \mathbf{c}_j \rangle < Ln^{k+1}$.* ∎

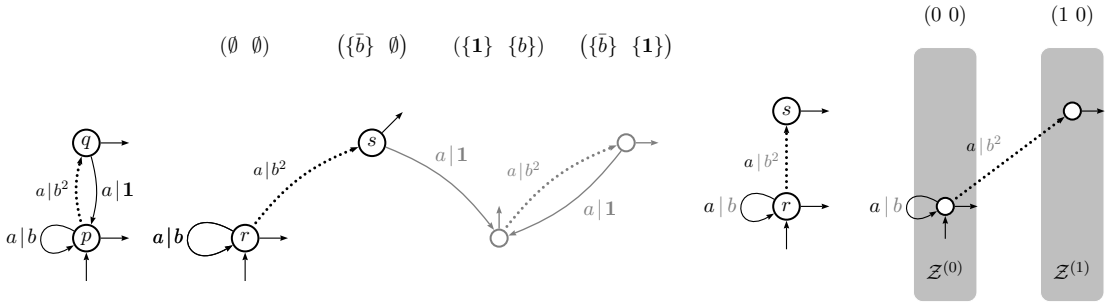
A concise proof for Lemma 4.3 can be derived from a property of the Lead or Delay action stated in Lemma 5 of [2]. Although not so long, it is omitted due to space constraints.

Proof of Proposition 4.2. Fix $N \geq Ln^{k+1}$. By Property 3.9, distinct successful computations of $\mathcal{V}_N$ with the same input either output distinct words or have a lag greater than Ln^{k+1}. Hence $k+1$ distinct successful computations of $\mathcal{V}_N$ with the same input word would project on a $(k+1)$-tuple of computations of $\mathcal{T}$ that contradicts Lemma 4.3. ∎

4.2. Decomposing the input-k-ambiguous transducer $\mathcal{V}_N$

As observed in Section 3.3, $\mathcal{T}$ and $\mathcal{V}_N$ are equivalent (for every N). Thus, a decomposition of $\mathcal{V}_N$ is also a decomposition of $\mathcal{T}$.

Take $N = Ln^{k+1}$ and let $\mathcal{A}$ be the underlying input automaton of $\mathcal{V}_N$. It is straightforward to decompose $\mathcal{V}_N$ by applying the multi-skimming covering on $\mathcal{A}$. By Proposition 4.2, $\mathcal{A}$ is k-ambiguous, hence the multi-skimming covering yields unambiguous automata $\mathcal{B}_k^{(0)}, \ldots, \mathcal{B}_k^{(k-1)}$ which are immersions in $\mathcal{A}$, and whose successful computations are in bijection with those of $\mathcal{A}$. By lifting to the transitions of $\mathcal{B}_k^{(0)}, \ldots, \mathcal{B}_k^{(k-1)}$ the corresponding outputs in $\mathcal{V}_N$ of the projected ones, we obtain unambiguous transducers $\mathcal{Z}^{(0)}, \ldots, \mathcal{Z}^{(k-1)}$ whose union is equivalent to $\mathcal{T}$. Figure 6 shows an example with a given ordering for each covering. Other decompositions are obtained by varying these orderings.



(a) A lag separation covering $\mathcal{U}_N$ with $N = 1$. The $\mathfrak{P}(\Delta_1)^Q$-vectors (vertical projection) are indexed by $\{p, q\}$, in that order. The dotted transition is larger than the solid one. The input-2-ambiguous (and equivalent to $\mathcal{T}$) subtransducer $\mathcal{V}_1$ is reduced to the states $\{r, s\}$.

(b) The lifted transducers $\mathcal{Z}^{(0)}$ and $\mathcal{Z}^{(1)}$ from a 2-skimming of the input automaton of $\mathcal{V}_1$. The choices of final states yield $|\mathcal{Z}^{(0)}|\colon a^n \mapsto b^n\,(n \geq 0)$ and $|\mathcal{Z}^{(1)}|\colon a^n \mapsto b^{n+1}\,(n > 0)$.

Figure 6: A decomposition of the 2-valued transducer $\mathcal{T}$ of Figure 2.

The number of states of the decomposition depends on the following parameters of $\mathcal{T}$: n (number of states), h (cardinality of the output alphabet), L (maximal of the lengths of the outputs of transitions) and k (valuedness). We claim:

Property 4.4. Each transducer $\mathcal{Z}^{(i)}$ has at most $2^{\mathcal{O}(hLk^4n^{k+4})}$ useful states.

The proof is based on a fine analysis of the useful states of $\mathcal{Z}^{(i)}$ and goes as follows. Let X be the set of useful states of $\mathcal{V}_N$ (as said in Section 3.3, $X \subseteq Q \times \mathfrak{P}_{(k)}(\Delta_N)^Q$). Each transducer $\mathcal{Z}^{(i)}$ is obtained by the multi-skimming covering of the underlying input automaton of $\mathcal{V}_N$, hence its states belong to $X \times \mathbb{N}_k^X$ (assuming that $\mathcal{Z}^{(i)}$ was built on the trim part of $\mathcal{V}_N$). By the stated properties of the constructions, we can derive that if[9] (P, V) is useful in $\mathcal{Z}^{(i)}$, then V has at most kn entries different from 0. In other words, the set of coordinates of V having a nonzero value belongs to $\mathfrak{P}_{(kn)}(X)$. There are k possible nonzero values for each such coordinate, namely $\{1, \ldots, k-1, \omega\}$, thus the number of useful states of $\mathcal{Z}_i$ is at most $\operatorname{card}(X) \cdot \operatorname{card}\left(\mathfrak{P}_{(kn)}(X)\right) \cdot k^{kn}$. To conclude, it remains to use the discussion on the number of useful states of $\mathcal{V}_N$ at the end of Section 3.3: we have that $\operatorname{card}\left(\mathfrak{P}_{(kn)}(X)\right) \leq \operatorname{card}(X)^{(kn)^2}$, and by Property 3.10, $\operatorname{card}(X) \leq 2^{2hNk^2n}$. With $N = n^{k+1}L$, we obtain the bound of $2^{\mathcal{O}(hLk^4n^{k+4})}$ states.

4.3. The morphic decomposition theorem

We turn now to Theorem 1.2, the proof of which goes in four steps. First, we construct a k-valued transducer $\mathcal{T}$ realising the composition $\tau\theta$. This is done by relabelling the transitions of the transducer $\mathcal{S}$ realising τ: every transition $p \xrightarrow{a|x} q$ of $\mathcal{S}$ is replaced by $p \xrightarrow{a|x\theta} q$. Next, $\mathcal{T}$ is decomposed into k unambiguous transducers $\mathcal{Z}^{(0)}, \ldots, \mathcal{Z}^{(k-1)}$. These transducers are immersions in $\mathcal{V}_N$ and, by composition of morphisms, also in $\mathcal{T}$; but it may be the case that not every successful computation of $\mathcal{T}$ is projected by some successful one in the union of the $\mathcal{Z}^{(i)}$. The third and crucial step (described more precisely below) consists, roughly speaking, to *stick* the successful computations of $\mathcal{T}$ to the transducers $\mathcal{Z}^{(0)}, \ldots, \mathcal{Z}^{(k-1)}$ in

[9]Capital letters are used in order to distinguish the states of $\mathcal{Z}^{(i)}$ from the states of other automata.

order to obtain equivalent (thus functional) transducers $\mathcal{W}^{(0)}, \ldots, \mathcal{W}^{(k-1)}$, not necessarily unambiguous, whose successful computations project on the whole set of successful computations of $\mathcal{T}$. Finally, the transitions of each $\mathcal{W}^{(i)}$ are relabelled in order to construct an immersion of $\mathcal{S}$: $e : p \xrightarrow{a|y} q$ in $\mathcal{W}^{(i)}$ projects on a transition f of $\mathcal{T}$; the label of f is, by construction, of form $a|x\theta$; the output y of e is replaced by x. This yields k transducers decomposing $\mathcal{S}$, not necessarily functional, but whose compositions with θ are functional.

The definition of the transducers $\mathcal{W}^{(0)}, \ldots, \mathcal{W}^{(k-1)}$ is based on a generalisation of the property of functional transducers that the lag between every pair of successful computations with same label is bounded by some integer (this appears implicitly in a proof of [2]).

Property 4.5. Let $N = n^{k+1}L$ and $K = 2(k+1)N$. If $\mathcal{T}$ is k-valued, then for every successful computation c of $\mathcal{T}$ there exists a successful computation D in $\mathcal{Z}^{(0)} \cup \cdots \cup \mathcal{Z}^{(k-1)}$ with same input, same output and such that $\langle c, D\varphi \rangle < K$. ∎

We can obtain each $\mathcal{W}^{(i)}$ from the product of $\mathcal{T} \times \mathcal{Z}^{(i)}$ by the Lead or Delay action, see [2] for details. The part of this product restricted to states having Lead or Delay in Δ_K projects on the successful computations of $\mathcal{T}$ with lag smaller than K with some successful computation in $\mathcal{Z}^{(i)}$. The number of states of $\mathcal{W}^{(i)}$ is bounded by $n \times M \times \mathrm{card}\,(\Delta_K)$, where M is the number of states of $\mathcal{Z}(i)$. This is again of order $2^{\mathcal{O}(hLk^4n^{k+4})}$.

References

[1] J. Berstel. *Transductions and Context-Free Languages*. B. G. Teubner, 1979.

[2] M.-P. Béal, O. Carton, C. Prieur, and J. Sakarovitch. Squaring transducers: an efficient procedure for deciding functionality and sequentiality. *Theoretical Computer Science*, 292:45–63, 2003.

[3] M.-P. Béal, S. Lombardy, and J. Sakarovitch. Conjugacy and equivalence of weighted automata and functional transducers. In D. Grigoriev, J. Harrison, and E. A. Hirsch, editors, *Proc. of CSR'06*, volume 3967 of *Lect. Notes in Comp. Science*, pp. 58–69, 2006.

[4] K. Culik and J. Karhumäki. The equivalence of finite valued transducers (on HDT0L languages) is decidable. *Theoretical Computer Science*, 47(1):71–84, 1986.

[5] S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, 1974.

[6] E. Gurari and O. Ibarra. A note on finite-valued and finitely ambiguous transducers. *Mathematical Systems Theory*, 16:61–66, 1983.

[7] J. H. Johnson. Do rational equivalence relations have regular cross-sections? In Wilfried Brauer, editor, *Proc. ICALP'85*, volume 194 of *Lect. Notes in Comp. Science*, pp. 300–309. Springer-Verlag, 1985.

[8] J. H. Johnson. Rational equivalence relations. *Theoretical Computer Science*, 47(3):39–60, 1986.

[9] J. Sakarovitch. A construction on finite automata that has remained hidden. *Theoretical Computer Science*, 204(1–2):205–231, 1998.

[10] J. Sakarovitch. *Éléments de théorie des automates*. Vuibert, 2003. English translation: *Elements of Automata Theory*, Cambridge University Press, to appear.

[11] J. Sakarovitch. The rational skimming theorem. In Do Long Van and M. Ito, editors, *Proc. of The Mathematical Foundations of Informatics (1999)*, World Scientific, pp. 157–172, 2005.

[12] J. Sakarovitch and R. de Souza. On the decidability of finite valuedness of transducers. in preparation (preliminary version available at `http://www.infres.enst.fr/~rsouza`).

[13] M. P. Schützenberger. Sur les relations rationnelles. In H. Barkhage, editor, *Automata Theory and Formal Languages, 2nd GI Conference*, volume 33 of *Lect. Notes in Comp. Science*, pp. 209–213, 1975.

[14] A. Weber. On the valuedness of finite transducers. *Acta Informatica*, 27(8):749–780, 1989.

[15] A. Weber. Decomposing a k-valued transducer into k unambiguous ones. *RAIRO Informatique Théorique et Applications*, 30(5):379–413, 1996.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 633-644
www.stacs-conf.org

THE ISOMORPHISM PROBLEM FOR PLANAR 3-CONNECTED GRAPHS IS IN UNAMBIGUOUS LOGSPACE

THOMAS THIERAUF [1] AND FABIAN WAGNER [2]

[1] Fak. Elektronik und Informatik, HTW Aalen, 73430 Aalen, Germany
E-mail address: thomas.thierauf@uni-ulm.de

[2] Inst. für Theoretische Informatik, Universität Ulm, 89069 Ulm, Germany
E-mail address: fabian.wagner@uni-ulm.de

Abstract. The isomorphism problem for planar graphs is known to be efficiently solvable. For planar 3-connected graphs, the isomorphism problem can be solved by efficient parallel algorithms, it is in the class $\mathbf{AC}^1$.

In this paper we improve the upper bound for planar 3-connected graphs to unambiguous logspace, in fact to $\mathbf{UL} \cap \mathbf{coUL}$. As a consequence of our method we get that the isomorphism problem for oriented graphs is in $\mathbf{NL}$. We also show that the problems are hard for $\mathbf{L}$.

1. Introduction

The graph isomorphism problem (GI) is one of the most challenging problems today. No polynomial time algorithm is known for it, even with extended resources like randomization or on quantum computers. On the other hand, it is not known to be $\mathbf{NP}$-complete and there are good reasons to conjecture that it is in fact not complete.

For some restricted classes of graphs, efficient algorithms for GI are known. For example for trees [AHU74] or for graphs with bounded degree [Luk82]. We are interested in planar graphs and 3-connected graphs. A graph is 3-connected if it remains connected after deleting two arbitrary vertices. In 1966, Weinberg [Wei66] presented an $O(n^2)$-algorithm for testing isomorphism of planar 3-connected graphs. This algorithm was improved and extended by Hopcroft and Tarjan [HT74] to an $O(n \log n)$-algorithm for the planar graph isomorphism problem (planar-GI). Then Hopcroft and Wong [HW74] showed that it is solvable in linear time. Since the constant hidden in the linear time bound is very large, the problem has been reconsidered under a more practical approach [KHC04]. The parallel complexity of planar-GI has been studied by Miller and Reif [MR91] and Ramachandran and Reif [RR94]. They showed that planar-GI $\mathbf{AC}^1$-reduces to the 3-connected case and that 3-connected GI is in $\mathbf{AC}^1$.

Grohe and Verbitsky [GV06] gave an alternative way to show that planar-GI is in $\mathbf{AC}^1$. They proved for a class $\mathcal{G}$ of graphs, that if every graph in $\mathcal{G}$ is definable in a finite-variable first order logic within logarithmic quantifier depth, then the isomorphism problem for $\mathcal{G}$

Supported by DFG grants Scho 302/7-2 and TO 200/2-1.

is in $\mathbf{AC}^1$. Later Verbitsky [Ver07] showed that planar 3-connected graphs are definable with 15 variables and quantifier depth $O(\log n)$ which leads to a 14-dimensional Weisfeiler-Lehman algorithm. With the reduction of [MR91] one obtains a new $\mathbf{AC}^1$-algorithm for planar-GI.

In the above papers on planar-GI, the authors consider first 3-connected graphs. The reason is a result due to Whitney [Whi33] that every planar 3-connected graph has precisely two embeddings on a sphere, where one embedding is the mirror image of the other. Moreover, one can efficiently compute these embeddings. Weinberg [Wei66] used these embeddings to compute a code for a graph, such that isomorphic graphs will have the same code. We call a code with this property a *canonical code* for the graph.

Some of the subroutines in the above algorithms have complexity below $\mathbf{AC}^1$. Allender and Mahajan [AM00] showed that planarity testing is hard for $\mathbf{L}$ and in symmetric logspace, $\mathbf{SL}$. Since $\mathbf{SL} = \mathbf{L}$ [Rei05], planarity testing is complete for logspace. Furthermore Allender and Mahajan [AM00] showed that a planar embedding can be computed in logspace. Also the connectivity structure of a (undirected) graph can be computed in logspace [NTS95]. Hence a natural question is whether planar-GI is in logspace.

While this question remains open, we considerably improve the upper bound for planar-GI for 3-connected graphs in Section 3, namely from $\mathbf{AC}^1$ to unambiguous logspace, in fact to $\mathbf{UL} \cap \mathbf{coUL}$. Like Weinberg, we construct codes for the given graphs. In order to use only logarithmic space, our code is constructed via a spanning tree, which depends on the planar embedding of the graph. A crucial tool in the construction of the spanning tree is based on a recent result by Bourke, Tewari, and Vinodchandran [BTV07] that the reachability problem for planar directed graphs is in $\mathbf{UL} \cap \mathbf{coUL}$. They built on work of Reinhard and Allender [RA00] and Allender, Datta, and Roy [ADR96]. We argue in Section 4 that their algorithm can be modified to not just solve reachability questions but to compute distances between nodes in $\mathbf{UL} \cap \mathbf{coUL}$.

The embedding of a planar graph can be represented as a *rotation scheme*. Intuitively this gives the edges in clockwise or counter clockwise order around each node such that it leads to a planar drawing of the graph. Rotation schemes have also been considered for non-planar graphs. We talk of *oriented graphs* in this case. We extend our results to the isomorphism problem for oriented graphs. There one has given two graphs G and H and a rotation scheme for each of the graphs. One has to decide whether there is an isomorphism between G and H that respects the rotation schemes. In Section 5 we show that the problem is in $\mathbf{NL}$.

With respect to lower bounds, GI is known to be hard for DET [Tor04], where DET is the class of problems that are $\mathbf{NC}^1$-reducible to the determinant defined by [Coo85]. In fact, already the isomorphism problem for tournament graphs is hard for DET [Wag07]. We show in Section 6 that the isomorphism problem for planar 3-connected graphs is hard for logspace.

2. Preliminaries

Basically, $\mathbf{L}$ and $\mathbf{NL}$ are the classes of languages computable by a deterministic and nondeterministic logspace bounded Turing machine, respectively. A nondeterministic Turing machine is called *unambiguous*, if it has at most one accepting computation on any

input. The class of languages computable by unambiguous logspace bounded Turing machines is denoted by **UL**. **NL** is known to be closed under complement [Imm88, Sze88], but it is open for **UL**.

The functional version of **L** is denoted by **FL**. It is known that **FL**-functions are closed under composition, i.e. $\mathbf{FL} \circ \mathbf{FL} = \mathbf{FL}$. The proof goes by recomputing bits of the function value of the first function each time such a bit is needed by the second function. The same argument works when we consider functions that are computed by unambiguous logspace bounded Turing machines. If we call the class **FUL**, then this says that $\mathbf{FUL} \circ \mathbf{FUL} = \mathbf{FUL}$. We need a further property of **UL**:

Lemma 2.1. $\mathbf{L}^{\mathbf{UL} \cap \mathbf{coUL}} = \mathbf{UL} \cap \mathbf{coUL}$.

Proof. Let M be a logspace oracle Turing machine with oracle $A \in \mathbf{UL} \cap \mathbf{coUL}$. Let M_0, M_1 be (nondeterministic) unambiguous logspace Turing machines such that $L(M_0) = \overline{A}$ and $L(M_1) = A$. An unambiguous logspace Turing machine M' for $L(M, A)$ works as follows on input x:

> Simulate M on input x. If M asks an oracle question y, then nondeterministically guess whether the answer is 0 or 1.
>
> - If the guess is answer 0, then simulate M_0 on input y. If M_0 accepts, then continue the simulation of M with oracle answer 0. If M_0 rejects then reject and halt.
> - If the guess is answer 1, then simulate M_1 on input y. If M_1 accepts, then continue the simulation of M with oracle answer 1. If M_1 rejects then reject and halt.
>
> Finally accept iff M accepts.

Note that M' is unambiguous because M_0 and M_1 are unambiguous and of the two guessed oracle answers always exactly one guess is correct. ∎

Let $G = (V, E)$ be an undirected graph with vertices $V = V(G)$ and edges $E = E(G)$. Let $G - \{v\}$ denote the induced subgraph of G on $V(G) \setminus \{v\}$. The *neighbours* of $v \in V$ are $\Gamma(v) = \{ u \mid (v, u) \in E \}$. By E_v we denote the edges going from v to its neighbors, $E_v = \{ (v, u) \mid u \in \Gamma(v) \}$. By $d(u, v)$ we denote the distance between nodes u and v in G, which is the length of a shortest path from u to v in G.

A graph is *connected* if there is a path between any two vertices in G. A vertex $v \in V$ is an *articulation point* if $G - \{v\}$ is not connected. A pair of vertices $u, v \in V$ is a *separation pair* if $G - \{u, v\}$ is not connected. A *biconnected graph* contains no articulation points. A 3-*connected graph* contains no separation pairs.

A *rotation scheme* for a graph G is a set ρ of permutations, $\rho = \{ \rho_v \mid v \in V \}$, where ρ_v is a permutation on E_v that has only one cycle (which is called a rotation). Let ρ^{-1} be the set of inverse rotations, $\rho^{-1} = \{ \rho_v^{-1} \mid v \in V \}$. A rotation scheme ρ describes an embedding of graph G in the plane. We call G together with ρ an *oriented graph*. If the embedding is planar, we call ρ a *planar rotation scheme*. Note that in this case ρ^{-1} is a planar rotation scheme as well. Allender and Mahajan [AM00] showed that a planar rotation scheme for a planar graph can be computed in logspace.

If a planar graph is in addition 3-connected, then there exist precisely two planar rotation schemes [Whi33], namely some planar rotation scheme ρ and its inverse ρ^{-1}. This is a crucial property in our isomorphism test.

3. Planar 3-Connected Graph Isomorphism

In this section we prove the following theorem.

Theorem 3.1. *The isomorphism problem for planar, 3-connected graphs is in* $\mathbf{UL} \cap \mathbf{coUL}$.

In 1966, Weinberg [Wei66] presented an $O(n^2)$ algorithm for testing isomorphism of planar 3-connected graphs. The algorithm computes a *canonical form* for each of the two graphs. This is a coding of graphs such that these codings are equal iff the two graphs are isomorphic. For a 3-connected graph G, the algorithm starts by constructing a code for every edge of G and any of the two rotation schemes. Of all these codes, the lexicographical smallest one is the code for G.

For a designated edge (s,t) and a rotation scheme ρ for G, the code is constructed roughly as follows. Every undirected edge is considered as two directed edges. Now one can define an Euler tour based on some rules for selecting the next edge. Basically, the rules distinguish between the case whether a vertex or edge was already visited or not. The next edge to consider is chosen to the left or right of the active edge according to ρ. Define edge (s,t) to be the start of the tour. The code consists of the nodes as they appear on the tour, where the names are replaced by the order of their first appearance on the tour. That is, the code starts with $(1,2)$ for the edge (s,t) and every later occurrence of s or t on the tour is replaced by 1 or 2, respectively.

Weinberg's algorithm doesn't work in logspace, because one has to store the vertices and edges already visited. We show how to construct a different code in **UL**. Let (s,t) be a designated edge and ρ be a rotation scheme for G. Our construction makes three steps.

(1) First we compute a canonical spanning tree T for G. This is a spanning tree which depends on (s,t), ρ, and G, but not on the way these inputs are represented.
(2) Next we construct a canonical list L of all edges of G. To do so, we traverse T and enumerate the edges of T and their neighbor edges according to ρ. The list L does not depend on the representation of G, ρ or T.
(3) Finally we rename the vertices depending on the position of their first occurrence in the list L and get a code word for G with respect to (s,t) and ρ.

We will see that the spanning tree in step 1 can be computed in (the functional version of) $\mathbf{UL} \cap \mathbf{coUL}$. The list and the renaming in step 2 and step 3 can be computed in logspace, **L**. Therefore the composition of the three steps is in $\mathbf{UL} \cap \mathbf{coUL}$.

The overall algorithm has to decide whether two given graphs G and H are isomorphic. To do so we fix (s,t) and ρ for G and cycle through all edges of H as designated edge and the two possible permutation schemes of H. Then G and H are isomorphic iff we find a code for H that matches the code for G. It is not hard to see that this outer loop is in logspace. Therefore the isomorphism test stays in $\mathbf{UL} \cap \mathbf{coUL}$.

Step 1: Construction of a canonical spanning tree

We show that the following problem can be solved in unambiguous logspace.

- *Input:* An undirected graph $G = (V,E)$, a rotation scheme ρ for G, and a designated edge $(s,t) \in E$.
- *Output:* A canonical spanning tree $T \subseteq E$ of G.

Recall that by a canonical spanning tree we mean that T does not depend on the input representation of ρ or G, any representation will result in the same spanning tree T.

The idea to construct the spanning tree is to traverse G with a breath-first search starting at node s. The neighbors of a node are visited in the order given by the rotation scheme ρ. Since the algorithm should work in logspace, we cannot afford to store all the nodes that we already visited, as in a standard breath-first search. We get around this problem by working with distances between nodes.

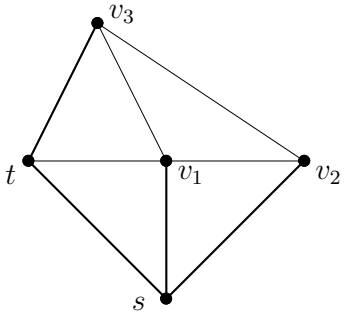
We start with the nodes at distance 1 from s. That is, write (s, v) on the output tape, for all $v \in \Gamma(s)$. Now let $d \geq 2$ and assume that we have already constructed T up to nodes at distance $\leq d-1$ to s. Then we consider the nodes at distance d from s. Let w be a node with $d(s, w) = d$. We have to connect w to the tree constructed so far. We do so by computing a shortest path from s to w. Ambiguities are resolved by using the first feasible edge according to ρ. We start with (s, t) as the active edge (u, v).

- If $d(u, w) > d(v, w)$, then (u, v) is the first edge encountered that is on a shortest path from u to w. Therefore we go from u to v and start searching the next edge from v. As starting edge we take the successor of (v, u). That is, $\rho_v(v, u)$ is the new active edge.
- If $d(u, w) \leq d(v, w)$, then (u, v) is not on a shortest path from u to w. Then we proceed with $\rho_u(u, v)$ as the new active edge.

After $d-1$ steps in direction of w the node v of the active edge (u, v) is a predecessor of w on a shortest path from s to w. Then we write (v, w) on the output tape. The following pseudo-code summarizes the algorithm.

for all $v \in \Gamma(s)$ **do output** (s, v)
for $d \leftarrow 2$ **to** $n-1$ **do**
  for all $w \in V$ such that $d(s, w) = d$ **do**
    $(u, v) \leftarrow (s, t)$
    for $k \leftarrow 1$ **to** $d-1$ **do**
      while $d(u, w) \leq d(v, w)$ **do** $(u, v) \leftarrow \rho_u(u, v)$
      $(u, v) \leftarrow \rho_v(v, u)$
    output (v, w)

The spanning tree T is canonical because its construction depends only on ρ, edge (s, t), and edge set E. The following figure shows an example of a spanning tree T for a graph G with rotation function ρ which arranges the edges in clockwise order around each vertex.



$$
\begin{aligned}
\rho &= \{\rho_s, \rho_t, \rho_{v_1}, \rho_{v_2}, \rho_{v_3}\} \\
\rho_s &= (\ (s,t)\ (s,v_1)\ (s,v_2)\ ) \\
\rho_t &= (\ (t,s)\ (t,v_3)\ (t,v_1)\ ) \\
\rho_{v_1} &= (\ (v_1,s)\ (v_1,t)\ (v_1,v_3)\ (v_1,v_2)\ ) \\
\rho_{v_2} &= (\ (v_2,s)\ (v_2,v_1)\ (v_2,v_3)\ ) \\
\rho_{v_3} &= (\ (v_3,t)\ (v_3,v_2)\ (v_3,v_1)\ )
\end{aligned}
$$

Except for the computation of the distances, the algorithm works in logspace. We have to store the values of d, k, u and v, and the position of w, plus some extra space for doing calculations. We show in Theorem 4.1 below that the distances can be computed in $\mathbf{UL} \cap \mathbf{coUL}$. By Lemma 2.1 the canonical spanning tree can be computed in $\mathbf{UL} \cap \mathbf{coUL}$.

Step 2: Computation of a canonical list of all edges

We show that the following problem can be solved in logspace.

- *Input:* An undirected graph $G = (V, E)$, a rotation scheme ρ for G, a spanning tree $T \subseteq E$ of G, and a designated edge $(s, t) \in T$.
- *Output:* A canonical list L of all edges in E.

Recall that by a canonical list we mean that the order of the edges as they appear in L does not depend on the input representation of ρ, G or T, any representation will result in the same list.

The idea is to traverse the spanning tree in a depth-first manner. At each vertex visit all incident edges in breath-first manner according to ρ until the next edge contained in the spanning tree is reached.

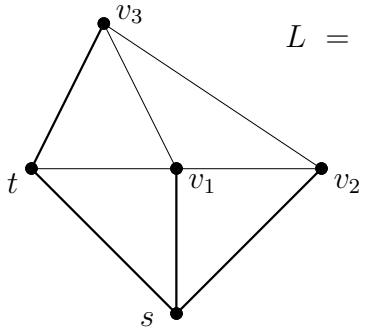
We start the traversal with edge (s, t) as the active edge (u, v). We write (u, v) on the output tape and then compute the next active edge as follows:

- If $(u, v) \in T$ then we walk depth-first in T from u to v, consider the edge (v, u) and take its successor according to ρ_v, i.e., $\rho_v(v, u)$ is the new active edge.
- If $(u, v) \notin T$ then we proceed breath-first with $\rho_u(u, v)$ as the new active edge.

This step is repeated until the active edge is again (s, t). Then we have traversed all edges in E. Every undirected edge is encountered exactly twice, once in each direction. The following pseudo-code summarizes the algorithm.

$(u, v) \leftarrow (s, t)$
repeat
  output (u, v)
  if $(u, v) \in T$ **then** $(u, v) \leftarrow \rho_v(v, u)$
  else $(u, v) \leftarrow \rho_u(u, v)$
until $(u, v) = (s, t)$

Clearly, the algorithm works in logspace. The list L is canonical because its construction depends only on ρ, edge (s, t), and sets E and T. Since T is canonical as well, L depends actually only on ρ, (s, t), and E. The following figure shows an example for L.



$$L = (s,t)(t,v_3)(v_3,v_2)(v_3,v_1)(v_3,t)(t,v_1)(t,s)$$
$$(s,v_1)(v_1,t)(v_1,v_3)(v_1,v_2)(v_1,s)$$
$$(s,v_2)(v_2,v_1)(v_2,v_3)(v_2,s)$$

Step 3: Renaming the vertices

The last step is to rename the vertices in the list L such that they become independent of the names they have in G. This is achieved as follows: consider the first occurrence (from left) of node v in L. Let $k - 1$ be the number of pairwise different nodes to the left of v. Then all occurrences of v are replaced by k. Recall that L starts with the edge (s, t). Hence

all occurrences of s get replaced by 1, all occurrences of t get replaced by 2, and so on. Call the new list $code(G, \rho, s, t)$.

Given L as input, the list $code(G, \rho, s, t)$ can be computed in logspace. We start with the first node v (which is s) and a counter k, that counts the number of different nodes we have seen so far. In the beginning, we set $k = 1$.

- If v occurs for the first time, than we output k and increase k by 1.
- If v occurs already to the left of the current position, then we have to determine the number, v got at its first occurrence. To do so, we determine the first occurance of v and then count the number of different nodes to the left of of v at its first occurance. It is not hard to see that this can be done in logspace.

Then we go to the next node in L. Consider the example from above. The code constructed from list L for G is as follows.

$L =$	(s,t)	(t,v_3)	(v_3,v_2)	(v_3,v_1)	(v_3,t)	(t,v_1)	(t,s)
$code(G,\rho,s,t) =$	$(1,2)$	$(2,3)$	$(3,4)$	$(3,5)$	$(3,2)$	$(2,5)$	$(2,1)$
sequel of L	(s,v_1)	(v_1,t)	(v_1,v_3)	(v_1,v_2)	(v_1,s)		
sequel of *code*	$(1,5)$	$(5,2)$	$(5,3)$	$(5,4)$	$(5,1)$		
sequel of L	(s,v_2)	(v_2,v_1)	(v_2,v_3)	(v_2,s)			
sequel of *code*	$(1,4)$	$(4,5)$	$(4,3)$	$(4,1)$			

The renaming algorithm works in logspace. It remains to argue that the new names of the nodes are independent of their names in G. Let H be a graph which is isomorphic to G, and let φ be an isomorphism between G and H. Note that $\rho \circ \varphi$ is a rotation scheme for H. Consider the computation of the code for graph H with rotation scheme $\rho \circ \varphi$ and designated edge $(\varphi(s), \varphi(t))$. The spanning tree computed in step 1 will be $\varphi(T)$ and the list computed in step 2 will be $\varphi(L)$. Now the above renaming procedure will give the same number to node v in L and to node $\varphi(v)$ in $\varphi(L)$. For example nodes $\varphi(s)$ and $\varphi(t)$ will get number 1 and 2, respectively. It follows that $code(G, \rho, s, t) = code(H, \rho \circ \varphi, \varphi(s), \varphi(t))$. We summarize:

Theorem 3.2. *Let G and H be connected, undirected graphs, let ρ_G be a rotation scheme for G and (s,t) be an edge in G. Then G and H are isomorphic iff there exists a rotation scheme ρ_H for H and an edge (u,v) in H such that $code(G, \rho_G, s, t) = code(H, \rho_H, u, v)$.*

This completes the proof of Theorem 3.1 except for the complexity bound on computing distances in planar graphs. This is done in the next section.

4. Computing Distances in Planar Graphs

We show that distances in planar graphs can be computed in unambiguous logspace.

Theorem 4.1. *The distance between any two vertices in a planar graph can be computed in* $\mathbf{UL} \cap \mathbf{coUL}$.

Bourke, Tewari, and Vinodchandran [BTV07] showed that the reachability problem for planar directed graphs is in $\mathbf{UL} \cap \mathbf{coUL}$. Their algorithm is essentially based on two results:

(1) Allender, Datta, and Roy [ADR96] showed that the reachability problem for planar directed graphs can be reduced to grid graph reachability. Grid graphs are graphs

who's vertices can be identified with the grid points in a 2-dimensional grid with the edges connecting only the direct horizontal or vertical neighbors.

(2) Reinhard and Allender [RA00] showed that the **NL**-complete reachability problem for directed graphs is in **UL**∩**coUL** if there is a logspace computable weight function for the edges such that for every pair of vertices u and v, if there is a path from u to v, then there is a unique minimum weight shortest path between u and v.

Bourke, Tewari, and Vinodchandran [BTV07] provide such a weight function for grid graphs. Therefore the reachability problem for planar directed graphs is in $\mathbf{UL} \cap \mathbf{coUL}$.

We modify this algorithm in order to determine distances between nodes in the given planar graph G. This is adapted from the Reinhard-Allender algorithm applied to the weighted grid graph computed from G. Here, we only describe the changes that have to be made in the cited references.

We start by considering the reduction from reachability for a planar graph G to a grid graph G_{grid} [ADR96]. The reduction from G to G_{grid} is a special combinatorial embedding that introduces only degree 2 nodes, thereby it preserves the exact number of paths between any two original vertices. Vertices in G are replaced by directed cycles and edges in G are replaced by paths such that they can be embedded into a grid. For our purpose it suffices to note that one can modify the construction and *mark the original edges of* G in G_{grid}. Hence if we consider paths in G_{grid} and count only the marked edges, we get distances in G.

The next step is to define a weight function such that shortest paths in G_{grid} with respect to marked edges are unique. Bourke, Tewari, and Vinodchandran [BTV07] defined the following weight function. For an edge e let

$$w_0(e) = \begin{cases} n^4, & \text{if } e \text{ is an east or west edge,} \\ n^4 + i, & \text{if } e \text{ is a north edge in column } i, \\ n^4 - i, & \text{if } e \text{ is a south edge in column } i. \end{cases}$$

Let p be a path in G_{grid}. The weight $w_0(p)$ is the sum of the weights of the edges on p and can be written as $a + bn^4$. Clearly, b is the number of edges on p. Also, it is easy to see that if another path p' with weight $w_0(p') = a' + b'n^4$ has the same weight as p, i.e.. $w_0(p) = w_0(p')$, then $a = a'$ and $b = b'$. This enforces that when we consider shortest paths between two nodes, these paths must have the same number of edges. The crucial part now is the value of a. Let p and p' be different simple paths connecting the same two vertices. Then Bourke, Tewari, and Vinodchandran [BTV07] showed that $a \neq a'$. It follows that the minimum weight path with respect to w_0 is always unique.

Now we modify the weight function in order to give priority to the marked edges. That is, we define

$$w(e) = \begin{cases} w_0(e) + n^8, & \text{if } e \text{ is marked,} \\ w_0(e), & \text{otherwise.} \end{cases}$$

Clearly, minimum weight paths must minimize the number of marked edges. The next parameter to minimize is the number of all edges on a path. Finally, by the same argument as above, the a-values of different simple paths that connect the same two vertices will be different. It follows that the minimum weight path with respect to w is always unique.

Reinhard and Allender [RA00] extended the counting technique of Immerman [Imm88] and Szelepcsényi [Sze88]. In addition to the number of nodes within distance k from some start node s, they also sum up the length of the shortest paths to these nodes. If the shortest paths are unique then they show that the predicate $d(s, v) \leq k$ is in $\mathbf{UL} \cap \mathbf{coUL}$. The

distance d now refers to G_{grid} because this is the input of the algorithm. By augmenting the algorithm with a counter for marked edges we also can refer to distances in G by construction of the weight function w. This suffices for our purpose because by several invocations of this procedure with different k's we can determine $d(s, v)$ for any s and v in $\mathbf{UL} \cap \mathbf{coUL}$, where d is the distance in G.

5. Oriented Graph Isomorphism

In the previous sections we have considered planar graphs, where the planar embedding is provided by a rotation scheme. It is also interesting to consider *arbitrary* (undirected) graphs with a rotation scheme that induces some orientation, i.e. cyclic order, on the edges. In the *isomorphism problem for oriented graphs* we have given two graphs, each with a rotation scheme. One has to decide whether there is an isomorphism between the graphs that respects the orientation.

Miller and Reif [MR91] proved that the isomorphism problem for oriented graphs is in $\mathbf{AC}^1$. We improve the complexity bound to **NL**. The proof goes along the same lines as for planar-GI: compute a canonical form for each of the graphs according to the given rotation schemes such that precisely in the isomorphic case, these canonical forms are equal.

Theorem 5.1. *The oriented graph isomorphism problem is in* **NL**.

It suffices to analyse the complexity of computing a canonical form for a graph G and a rotation scheme ρ. If G is not connected, then we determine the connected components in logspace [NTS95, Rei05] and compute canonical forms for each of them. Then we sort these canonical forms lexicographically and write them onto the output tape. Thus, we may assume that G is connected.

The three steps to compute a canonical form for a planar graph were all in logspace, except for the subroutine to compute distances, which was in $\mathbf{UL} \cap \mathbf{coUL}$. Without planarity, the best upper bound for computing distances in a graph is **NL**: to determine if $d(u, v) \leq k$ simply guess a path of length $\leq k$ from u to v. This proves Theorem 5.1.

6. Hardness of Planar 3-Connected GI

Lindell [Lin92] proved that tree isomorphism (TI) is in **L**. In fact, TI is complete for **L** [JKMT03]. Since trees are planar graphs, planar-GI is hard for **L**. We show that the problem remains hard for **L** even when restricted to planar 3-connected graphs. All the hardness and completeness results in this section are with respect to $\mathbf{AC}^0$-many-one reductions.

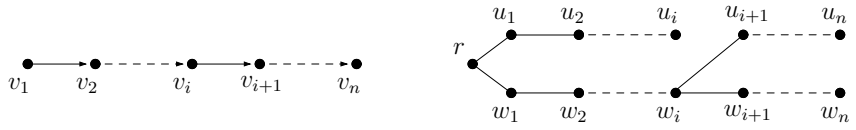
Theorem 6.1. *Planar 3-connected graph isomorphism is hard for* **L**.

We reduce from the known **L**-complete problem ORD which is defined as follows.

Order between Vertices (ORD)
Input: a directed graph $G = (V, E)$ that is a line, and $s, t \in V$.
Decide whether $s < t$ in the total order induced on V by G.

We first describe the reduction from ORD to TI [JKMT03]. Let $v_1, \ldots, v_n$ be the nodes of G in the order they appear on the line in G. In particular, v_1 is the unique node with in-degree 0 and v_n is the unique node with out-degree 0. Let $s = v_i$ and $t = v_j$. W.l.o.g. assume that $i \neq n$ (otherwise map the instance to a non-isomorphic pair of trees). The

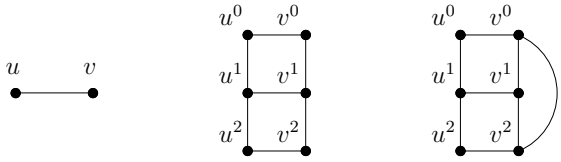
(undirected) tree T constructed from G has two copies $u_1, \ldots, u_n$ and $w_1, \ldots, w_n$ of the line of G, and there is an additional node r that is connected to u_1 and w_1. Up to this point, we have constructed one long line. Now the trick is to interrupt this line: take out the edge (u_i, u_{i+1}) and instead put the edge (w_i, u_{i+1}). Let T be the resulting tree.



G T

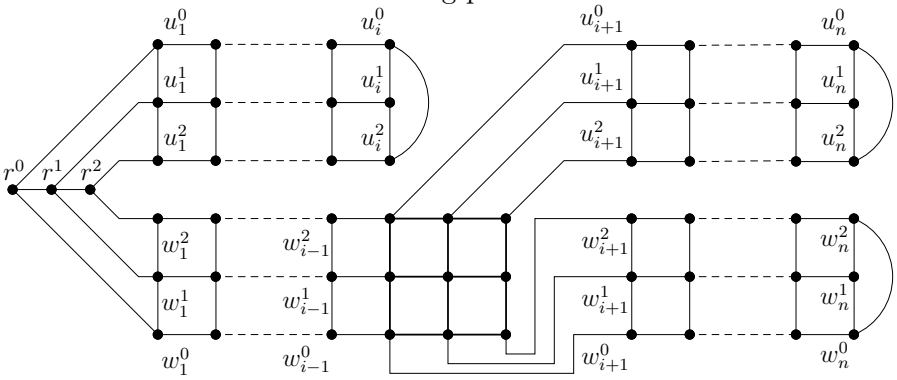
Note that there is a unique non-trivial automorphism for T: exchange u_{i+1} and w_{i+1}, $\ldots$, u_n and w_n, and map the other vertices onto themselves. We construct two trees T_1 and T_2 from T. With respect to T, tree T_1 has two extra nodes x_0, x_1 which are connected with node u_j, and T_2 has extra nodes y_0, y_1 which are connected with node w_j. The extra edges enforce that an isomorphism between T_1 and T_2 has to map u_j to w_j, because these are the only nodes of degree 4 (for $j < n$). Now, if $v_i < v_j$, then the above automorphism of T yields an isomorphism between T_1 and T_2. On the other hand, if $v_i \geq v_j$, then there is no isomorphism between T_1 and T_2.

We modify T to a graph H that is no longer a tree, but planar and 3-connected. Split each node v of degree 1 or 2 in T into three nodes v^0, v^1, v^2. Connect these nodes via edges (v^0, v^1) and (v^1, v^2). If v has degree 1, then additionally put the edge (v^0, v^2). Now, if (u, v) is an edge in T, where u and v have degree 1 or 2, then we have edges (u^0, v^0), (u^1, v^1), and (u^2, v^2) in H. The following picture illustrates the situation. In (a), node v has degree 2, in (b), node v has degree 1.



(a) (b)

A special case is node w_i which has degree 3. For w_i we need a gadget with 9 nodes which are connected as a 3×3 grid. The connections from this graph gadget (bold lines) to the other nodes are shown in the following picture.



H

Now it suffices again to mark the nodes corresponding to v_j. That is, define graph H_1 as graph H plus the edge (u_j^0, u_j^2), and H_2 as H plus the edge (w_j^0, w_j^2). Note that H_1 and H_2 are planar and 3-connected. Furthermore, any isomorphism between H_1 and H_2 has to map u_j^0 to w_j^0, u_j^1 to w_j^1, and u_j^2 to w_j^2. Again, this is only possible iff $v_i < v_j$. This completes the proof of Theorem 6.1.

A final observation is about oriented trees. An *oriented tree* is a tree with a planar rotation scheme. It is not hard to see that one can adapt Lindell's algorithm to work for oriented trees, so that the corresponding isomorphism problem is in **L**. We show that it is also hard for **L**.

Theorem 6.2. *Oriented tree isomorphism is complete for* **L**.

We reduce ORD to the oriented tree isomorphism problem. Let G be the given line graph and consider again the trees T_1 and T_2 from above constructed from G in the proof of Theorem 6.1. For nodes of degree 1 or 2 there is only one rotation scheme. Therefore we only have to take care of the nodes of degree 3 and 4, i.e. w_i, w_j, and u_j.

- The rotation scheme for w_i is easy to handle: output the edges around w_i for T_1 in an arbitrary order, and choose the opposite order for w_i in T_2. This definition fits together with the only possible isomorphism that should exchange u_{i+1} and w_{i+1}.
- In the rotation scheme for w_j the order of edges to the neighbors can be chosen as w_{j-1}, y_0, w_{j+1}, y_1, and around u_j in order u_{j-1}, x_0, u_{j+1}, x_1. Because of the symmetry of the parts (u_j, x_0) and (u_j, x_1) in T_1 and of (w_j, y_0) and (w_j, y_1) in T_2 an isomorphism mapping w_j to u_j can be defined respecting the rotation schemes for these nodes.

Now the same argument as for Theorem 6.1 shows that the oriented trees T_1 and T_2 are isomorphic iff $v_i < v_j$. This proves the theorem.

Open Problems

The most challenging task is to close the gap between **L** and $\mathbf{UL} \cap \mathbf{coUL}$ for the planar 3-connected graph isomorphism problem. Another goal is to extend the isomorphism test to arbitrary planar graphs. If the graph is not connected, we can compute the connected components and consider them separately. Hence, we may assume that the graph is connected. Then we can determine the articulation points and the separating pairs and get the 1- and 2-connected components of the graph. For sequential algorithms to compute a canonical form for these graphs see for example [KHC04]. Miller and Reif [MR91] provide an $\mathbf{AC}^1$-reduction from planar graphs to planar 3-connected graphs. We ask whether one can compute a canonical form for planar graphs in (unambiguous) logspace.

Acknowledgment

We thank Jacobo Torán and the anonymous referees for helpful comments on the manuscript.

References

[ADR96] E. Allender, S. Datta, and S. Roy. The directed planar reachability problem. In *Proc. 16th FST&TCS*, Lect. Notes in Comp. Science 1180, pp. 322–334, Springer, 1996.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass, 1974.

[AM00] E. Allender and M. Mahajan. The complexity of planarity testing. In *Proc. 17th STACS*, Lect. Notes in Comp. Science 1770, pp. 87–98, Springer, 2000.

[BTV07] C. Bourke, R. Tewari, and N.V. Vinodchandran. Directed planar reachability is in unambiguous logspace. In *22nd Annual IEEE Conference on Computational Complexity*, pp. 217–221, 2007.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[GV06] M. Grohe and O. Verbitsky. Testing graph isomorphism in parallel by playing a game. In *Proc. 33rd ICALP*, Lect. Notes in Comp. Science 4051, pp. 3–14, Springer, 2006.

[HT74] J.E. Hopcroft and R.E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21, 1974.

[HW74] J.E. Hopcroft and J.K. Wong. Linear time algorithm for isomorphism of planar graphs. In *6th ACM Symposium on Theory of Computing (STOC)*, 1974.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17:935–938, 1988.

[JKMT03] B. Jenner, J. Köbler, P. McKenzie, and J. Torán. Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, 66:549–566, 2003.

[KHC04] J.P. Kukluk, L.B. Holder, and D.J. Cook. Algorithm and experiments in testing planar graphs for isomorphism. *Journal of Graph Algorithms and Applications*, 8(2), 2004.

[Lin92] S. Lindell. A logspace algorithm for tree canonization (extended abstract). In *34th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 400–404. ACM, 1992.

[Luk82] E.M. Luks. Isomorphism of graphs of bounded valance can be tested in polynomial time. *Journal of Computer and System Sciences*, 25, 1982.

[MR91] G.L. Miller and J.H. Reif. Parallel tree contraction, part 2: Further applications. *SIAM Journal on Computing*, 20, 1991.

[NTS95] N. Nisan and A. Ta-Shma. Symmetric logspace is closed under complement. *Chicago Journal of Theoretical Computer Science*, 1995(Article 1), 1995.

[RA00] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM Journal on Computing*, 29(4):1118–1131, 2000.

[Rei05] O. Reingold. Undirected ST-connectivity in log-space. In ACM, editor, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing(STOC)*, pp. 376–385. ACM Press, 2005.

[RR94] V. Ramachandran and J.H. Reif. Planarity testing in parallel. *Journal of Computer and System Sciences*, 49, 1994.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[Tor04] J. Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing (SICOMP)*, 33(5):1093–1108, 2004.

[Ver07] O. Verbitsky. Planar graphs: Logical complexity and parallel isomorphism tests. In *24th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pp. 682–693, 2007.

[Wag07] F. Wagner. Hardness results for tournament isomorphism and automorphism. In *Proc. 32nd MFCS*, Lect. Notes in Comp. Science 4708, pp. 572–583, Springer, 2007.

[Wei66] H. Weinberg. A simple and efficient algorithm for determining isomorphism of planar triply connected graphs. *Circuit Theory*, 13:142–148, 1966.

[Whi33] H. Whitney. A set of topological invariants for graphs. *Amer. J. Mathematics*, 55:321–235, 1933.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 645-656
www.stacs-conf.org

EFFICIENT MINIMIZATION OF DFAS WITH PARTIAL TRANSITION FUNCTIONS

ANTTI VALMARI [1] AND PETRI LEHTINEN [1]

[1] Tampere University of Technology, Institute of Software Systems, PO Box 553, FI-33101 Tampere, Finland
E-mail address: {Antti.Valmari,Petri.Lehtinen}@tut.fi

ABSTRACT. Let *PT-DFA* mean a deterministic finite automaton whose transition relation is a partial function. We present an algorithm for minimizing a PT-DFA in $O(m \lg n)$ time and $O(m + n + \alpha)$ memory, where n is the number of states, m is the number of *defined* transitions, and α is the size of the alphabet. Time consumption does not depend on α, because the α term arises from an array that is accessed at random and never initialized. It is not needed, if transitions are in a suitable order in the input. The algorithm uses two instances of an array-based data structure for maintaining a refinable partition. Its operations are all amortized constant time. One instance represents the classical blocks and the other a partition of transitions. Our measurements demonstrate the speed advantage of our algorithm on PT-DFAs over an $O(\alpha n \lg n)$ time, $O(\alpha n)$ memory algorithm.

1. Introduction

Minimization of a deterministic finite automaton (DFA) is a classic problem in computer science. Let n be the number of states, m the number of transitions and α the size of the alphabet of the DFA. Hopcroft made a breakthrough in 1970 by presenting an algorithm that runs in $O(n \lg n)$ time, treating α as a constant [5]. Gries made the dependence of the running time of the algorithm on α explicit, obtaining $O(\alpha n \lg n)$ [3]. (Complexity is reported using the RAM machine model under the uniform cost criterion [1, p. 12].)

Our starting point was the paper by Knuutila in 2001, where he presented yet another $O(\alpha n \lg n)$ algorithm, and remarked that some versions which have been believed to run within this time bound actually fail to do so [6]. Hopcroft's algorithm is based on using only the "smaller" half of some set (known as *block*) that has been split. Knuutila demonstrated with an example that although the most well-known notion of "smaller" automatically leads to $O(\alpha n \lg n)$, two other notions that have been used may yield $\Omega(n^3)$ when $\alpha = \frac{1}{2}n$. He also showed that this can be avoided by maintaining, for each symbol, the set of those states in the block that have input transitions labelled by that symbol. According to [3], Hopcroft's original algorithm did so. Some later authors have dropped this complication as unnecessary, although it is necessary when the alternative notions of "smaller" are used.

Key words and phrases: deterministic finite automaton, sparse adjacency matrix, partition refinement.
Petri Lehtinen was funded by Academy of Finland, project ALEA (210795).

Knuutila mentioned as future work whether his approach can be used to develop an $O(m \lg n)$ algorithm for DFAs whose transition functions are not necessarily total. For brevity, we call them PT-DFAs. With an ordinary DFA, $O(m \lg n)$ is the same as $O(\alpha n \lg n)$ as $m = \alpha n$, but with a PT-DFA it may be much better. We present such an algorithm in this paper. We refined Knuutila's method of maintaining sets of states with relevant input transitions into a full-fledged data structure for maintaining refinable partitions. Instead of maintaining those sets of states, our algorithm maintains the corresponding sets of transitions. Another instance of the structure maintains the blocks.

Knuutila seems to claim that such a PT-DFA algorithm arises from the results in [7], where an $O(m \lg n)$ algorithm was described for refining a partition against a relation. However, there $\alpha = 1$, so the solved problem is not an immediate generalisation of ours. Extending the algorithm to $\alpha > 1$ is not trivial, as can be appreciated from the extension in [2]. It discusses $O(m \lg n)$ without openly promising it. Indeed, its analysis treats α as a constant. It seems to us that its running time does have an αn term.

In Section 2 we present an abstract minimization algorithm that, unlike [3, 6], has been adapted to PT-DFAs and avoids scanning the blocks and the alphabet in nested loops. The latter is crucial for converting αn into m in the complexity. The question of what blocks are needed in further splitting, has led to lengthy and sometimes unconvincing discussions in earlier literature. Our correctness proof deals with this issue using the "loop invariant" paradigm advocated in [4]. Our loop invariant "knows" what blocks are needed.

Section 3 presents an implementation of the refinable partition data structure. Its performance relies on a carefully chosen combination of simple low-level programming details.

The implementation of the main part of the abstract algorithm is the topic of Section 4. The analysis of its time consumption is based on proving of two lines of the code that, whenever the line is executed again for the same transition, the end state of the transition resides in a block whose size is at most half the size in the previous time. The numbers of times the remaining lines are executed are then related to these lines.

With a time bound as tight as ours, the order in which the transitions are presented in the input becomes significant, since the $\Theta(m \lg m)$ time that typical good sorting algorithms tend to take does not necessarily fit $O(m \lg n)$. We discuss this problem in Section 5, and present a solution that runs in $O(m)$ time but may use more memory, namely $O(m + \alpha)$.

Some measurements made with our implementations of Knuutila's and our algorithm are shown in Section 6.

2. Abstract Algorithm

A *PT-DFA* is a 5-tuple $\mathcal{D} = (Q, \Sigma, \delta, \hat{q}, F)$ such that Q and Σ are finite sets, $\hat{q} \in Q$, $F \subseteq Q$ and δ is explained below. The elements of Q are called *states*, $\hat{q}$ is the *initial state*, and F is the set of *final states*. The set Σ is the *alphabet*. We have $\delta \subseteq Q \times \Sigma \times Q$, and δ satisfies the condition that if $(q, a, q_1) \in \delta$ and $(q, a, q_2) \in \delta$, then $q_1 = q_2$. The elements of δ are *transitions*. In essence, δ is a partial function from $Q \times \Sigma$ to Q. Therefore, if $(q, a, q') \in \delta$, we write $\delta(q, a) = q'$. If $q \in Q$ and $a \in \Sigma$ but there is no q' such that $(q, a, q') \in \delta$, we write $\delta(q, a) = \perp$, where $\perp$ is some symbol satisfying $\perp \notin Q$. We will use $|\delta|$ as the number of transitions, and this number may be much smaller than $|Q||\Sigma|$, which is the number of transitions if δ is a full function.

By $q -a_1 a_2 \cdots a_n\rightarrow q'$ we denote that there is a path from state q to state q' such that the labels along the path constitute the word $a_1 a_2 \cdots a_n$. That is, $q -\varepsilon\rightarrow q$ holds for

```
1  (Q, Σ, δ, q̂, F) := remove irrelevant states and transitions from (Q, Σ, δ, q̂, F)
2  if F = ∅ then return empty_DFA(Σ)
3  else
4     if Q = F then B := {F} else B := {F, Q − F}
5     U := { (B, a) | B ∈ B ∧ a ∈ Σ ∧ δ_{B,a} ≠ ∅ }
6     while U ≠ ∅ do
7        (B, a) := any_element_of(U); U := U − {(B, a)}
8        for C ∈ B such that ∃q ∈ C : δ(q, a) ∈ B do
9           C_1 := { q ∈ C | δ(q, a) ∈ B }; C_2 := C − C_1
10          if C_2 ≠ ∅ then
11             B := B − {C}; B := B ∪ {C_1, C_2}
12             if |C_1| ≤ |C_2| then small := 1; big := 2 else small := 2; big := 1
13             U := U ∪ { (C_small, b) | δ_{C_small,b} ≠ ∅ ∧ b ∈ Σ }
14             U := U ∪ { (C_big, b) | δ_{C_big,b} ≠ ∅ ∧ (C, b) ∈ U }
15             U := U − ( {C} × Σ )
16    Q' := B; δ' := ∅; q̂' := Block(q̂); F' := ∅
17    for B ∈ B do
18       q := any_element_of(B)
19       if q ∈ F then F' := F' ∪ {B}
20       for a ∈ Σ such that δ(q, a) ≠ ⊥ do δ' := δ' ∪ { (B, a, Block(δ(q, a))) }
21    return (Q', Σ, δ', q̂', F')
```

Figure 1: Abstract PT-DFA minimization algorithm

every $q \in Q$, and $q -a_1a_2\cdots a_na_{n+1}\rightarrow q'$ holds if and only if there is some $q'' \in Q$ such that $q -a_1a_2\cdots a_n\rightarrow q''$ and $\delta(q'', a_{n+1}) = q' \neq \bot$. The *language* accepted by $\mathcal{D}$ is the set of words labelling the paths from the initial state to final states, that is, $\mathcal{L}(\mathcal{D}) = \{\, \sigma \in \Sigma^* \mid \exists q \in F : \hat{q} -\sigma\rightarrow q \,\}$. We will also talk about the languages of individual states, that is, $\mathcal{L}(q) = \{\, \sigma \in \Sigma^* \mid \exists q' \in F : q -\sigma\rightarrow q' \,\}$. Obviously $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\hat{q})$.

We say that a state is *relevant*, if and only if either it is the initial state, or it is reachable from the initial state and some final state is reachable from it. More precisely, $R = \{\hat{q}\} \cup \{\, q \in Q \mid \exists q' \in F : \exists \sigma \in \Sigma^* : \exists \rho \in \Sigma^* : \hat{q} -\sigma\rightarrow q -\rho\rightarrow q' \,\}$. It is obvious that irrelevant states and their adjacent transitions may be removed from a PT-DFA without affecting its language. The initial state cannot be removed, because otherwise the result would violate the condition $\hat{q} \in Q$ in the definition of a DFA. The removal yields the PT-DFA $(R, \Sigma, \delta', \hat{q}, F')$, where $\delta' = \delta \cap (R \times \Sigma \times R)$ and $F' = F \cap R$.

If no final state is reachable from the initial state, then $\mathcal{L}(\mathcal{D}) = \emptyset$. This is handled as a special case in our algorithm, because otherwise the result might contain unnecessary transitions from the initial state to itself. For this purpose, let empty_DFA(Σ) be $(\{x\}, \Sigma, \emptyset, x, \emptyset)$, where x is just any element. Obviously empty_DFA(Σ) is the smallest PT-DFA with the alphabet Σ that accepts the empty language.

The abstract minimization algorithm is shown in Figure 1. In it, $\mathcal{B}$ denotes a *partition* on Q. That is, $\mathcal{B}$ is a collection $\{B_1, B_2, \ldots, B_n\}$ of nonempty subsets of Q such that $B_1 \cup B_2 \cup \cdots \cup B_n = Q$, and $B_i \cap B_j = \emptyset$ whenever $1 \leq i < j \leq n$. The elements of $\mathcal{B}$ are called *blocks*. By checking all statements that modify the contents of $\mathcal{B}$, it is easy to verify that after its initialization on line 4, $\mathcal{B}$ is a partition on Q throughout the execution of the algorithm, except temporarily in the middle of line 11.

By $Block(q)$ we denote the block to which state q belongs. Therefore, if $q \in Q$, then $q \in Block(q) \in \mathcal{B}$. For convenience, we define $Block(\bot) = \bot \notin \mathcal{B}$. If $Block(q_1) \neq Block(q_2)$ ever starts to hold, then it stays valid up to the end of the execution of the algorithm.

Elements of $\mathcal{B} \times \Sigma$ are called *splitters*. Let $\delta_{B,a} = \{\, (q, a, q') \in \delta \mid q' \in B \,\}$. We say that splitter (B, a) is *nonempty*, if and only if $\delta_{B,a} \neq \emptyset$. The set $\mathcal{U}$ contains those nonempty splitters that are currently "unprocessed". It is obvious from line 8 that empty splitters would have no effect. The main loop of the algorithm (lines 6...15) starts with all nonempty splitters as unprocessed, and ends when no nonempty splitter is unprocessed. The classic algorithm uses either only F or only $Q - F$ for constructing the initial splitters, but this does not work with a partial δ.

The goal of the main loop is to split blocks until they are consistent with δ, without splitting too much. We will now prove in two steps that this is achieved.

Lemma 2.1. *For every $q_1 \in Q$ and $q_2 \in Q$, if $Block(q_1) \neq Block(q_2)$ at any time of the execution of the algorithm in Figure 1, then $\mathcal{L}(q_1) \neq \mathcal{L}(q_2)$.*

Proof. If the algorithm puts states q_1 and q_2 into different blocks on line 4, then either $\varepsilon \in \mathcal{L}(q_1) \wedge \varepsilon \notin \mathcal{L}(q_2)$ or $\varepsilon \notin \mathcal{L}(q_1) \wedge \varepsilon \in \mathcal{L}(q_2)$. Otherwise, it does so on line 11. Then there are i, j, B and a such that $\{i, j\} = \{1, 2\}$, $\delta(q_i, a) \in B$ and $\delta(q_j, a) \notin B$. Let $q'_i = \delta(q_i, a)$.

If $\delta(q_j, a) \neq \bot$, then let $q'_j = \delta(q_j, a)$. We have $q'_j \notin B$. Because the algorithm has already put q'_i and q'_j into different blocks (they were in different blocks on line 9), there is some $\sigma \in \Sigma^*$ such that either $\sigma \in \mathcal{L}(q'_i) \wedge \sigma \notin \mathcal{L}(q'_j)$ or vice versa. As a consequence, $a\sigma$ is in $\mathcal{L}(q_1)$ or in $\mathcal{L}(q_2)$, but not in both.

Assume now that $\delta(q_j, a) = \bot$. Because of lines 1 and 2, $\mathcal{L}(q) \neq \emptyset$ for every $q \in Q$. There is thus some $\sigma \in \Sigma^*$ such that $\sigma \in \mathcal{L}(q'_i)$. We have $a\sigma \in \mathcal{L}(q_i)$. Clearly $a\sigma \notin \mathcal{L}(q_j)$. ∎

At this point it is worth noticing that line 1 is important for the correctness of the algorithm. Without it, there could be two reachable states q_1 and q_2 that accept the same language, and a such that $\delta(q_1, a) = \bot$ while $\delta(q_2, a)$ is a state that accepts the empty language. The algorithm would eventually put q_1 and q_2 into different blocks.

We have shown that the main loop does not split blocks when it should not. We now prove that it splits all the blocks that it should.

Lemma 2.2. *At the end of the algorithm in Figure 1, for every $q_1 \in Q$, $q_2 \in Q$ and $a \in \Sigma$, if $Block(q_1) = Block(q_2)$, then $Block(\delta(q_1, a)) = Block(\delta(q_2, a))$.*

Proof. To improve readability, let $B_1 = Block(\delta(q_1, a))$ and $B_2 = Block(\delta(q_2, a))$. In the proof, $Block()$, B_1 and B_2 are always evaluated with the current $\mathcal{B}$, so their contents change. The proof is based on the following loop invariant:

> On line 6, for every $q_1 \in Q$, $q_2 \in Q$ and $a \in \Sigma$, if $Block(q_1) = Block(q_2)$, then $B_1 = B_2$ or $(B_1, a) \in \mathcal{U}$ or $(B_2, a) \in \mathcal{U}$.

Consider the situation immediately after line 5. If $B_1 \neq \bot$, then $(B_1, a) \in \mathcal{U}$. If $B_2 \neq \bot$, then $(B_2, a) \in \mathcal{U}$. If $B_1 = B_2 = \bot$, then $B_1 = B_2$. Thus the invariant holds initially.

Consider any q_1, q_2, a and instance of executing line 6 such that the invariant holds. Our task is to show that the invariant holds for them also when line 6 is executed for the next time.

The case that the invariant holds because $Block(q_1) \neq Block(q_2)$ is simple. Blocks are never merged, so $Block(q_1) \neq Block(q_2)$ is valid also the next time.

Consider the case $Block(q_1) = Block(q_2)$, $B_1 \neq B_2$ and $(B_i, a) \in \mathcal{U}$, where $i = 1$ or $i = 2$. Let $j = 3 - i$. If (B_i, a) is the (B, a) of line 7, then, when $Block(q_i)$ is the C of the

for-loop, q_i goes to C_1 and q_j goes to C_2. So $Block(q_1) = Block(q_2)$ ceases to hold, rescuing the invariant. If (B_i, a) is not the (B, a) of line 7, then, whenever B_i is split, lines 13 and 14 take care that both halves end up in $\mathcal{U}$. Thus $(B_i, a) \in \mathcal{U}$ stays true keeping the invariant valid, although B_i and $\mathcal{U}$ may change.

Let now $Block(q_1) = Block(q_2)$ and $B_1 = B_2$. To invalidate the invariant, B_1 or B_2 must be changed so that $B_1 = B_2$ ceases to hold. When this happens, line 13 puts (B_i, a) into $\mathcal{U}$, where $i = 1$ or $i = 2$. Like above, lines 13 and 14 keep (B_i, a) in $\mathcal{U}$ although B_i may change until line 6 is entered again.

We have completed the proof that the invariant stays valid.

When line 16 is entered, $\mathcal{U} = \emptyset$. The invariant now yields that if $Block(q_1) = Block(q_2)$, then $Block(\delta(q_1, a)) = Block(\delta(q_2, a))$. ∎

It is not difficult to check that lines 16...20 yield a PT-DFA, that is, Q' and Σ are finite sets and so on. In particular, the construction gives $\delta'(B, a)$ a value at most once. We now show that the result is the right PT-DFA.

Lemma 2.3. *Let $\mathcal{D}' = (Q', \Sigma, \delta', \hat{q}', F')$ be the result of the algorithm in Figure 1. We have $\mathcal{L}(\mathcal{D}') = \mathcal{L}(\mathcal{D})$. Furthermore, every PT-DFA that accepts $\mathcal{L}(\mathcal{D})$ has at least as many states and transitions as $\mathcal{D}'$. If it has the same number of states, it is either isomorphic with $\mathcal{D}'$ (ignoring Σ in the comparison), or it is of the form $(\{\hat{q}''\}, \Sigma'', \delta'', \hat{q}'', \emptyset)$ with $\delta'' \neq \emptyset$.*

Proof. The case where the algorithm exits on line 2 is trivial and has been discussed, so from now on we discuss the case where the algorithm goes through the main part.

Let $q \in Q$ and $a \in \Sigma$. Lemma 2.2 implies that $Block(\delta(q, a)) = Block(\delta(q', a))$ for every $q' \in Block(q)$. From this line 20 yields $\delta'(Block(q), a) = Block(\delta(q, a))$. By induction, if $\sigma \in \Sigma^*$, $q' \in Q$ and $q -\sigma\rightarrow q'$ in $\mathcal{D}$ then $Block(q) -\sigma\rightarrow Block(q')$ in $\mathcal{D}'$, and if $Block(q) -\sigma\rightarrow B \neq \perp$ in $\mathcal{D}'$ then there is $q' \in Q$ such that $B = Block(q')$ and $q -\sigma\rightarrow q'$ in $\mathcal{D}$. Similarly, lines 4 and 19 guarantee that $q' \in F$ if and only if $Block(q') \in F'$. Together these yield $\mathcal{L}(q) = \mathcal{L}(Block(q))$ and, in particular, $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\hat{q}) = \mathcal{L}(\hat{q}') = \mathcal{L}(\mathcal{D}')$.

Let $(Q'', \Sigma, \delta'', \hat{q}'', F'')$ be any PT-DFA that accepts the same language as $\mathcal{D}'$. Let $q' \in Q'$. Because the algorithm executed the main part, there are some $\sigma \in \Sigma^*$ and $\rho \in \Sigma^*$ such that $\hat{q}' -\sigma\rightarrow q'$ and $\rho \in \mathcal{L}(q')$. So $\sigma\rho \in \mathcal{L}(\hat{q}') = \mathcal{L}(\hat{q}'')$, and also Q'' contains a state q'' such that $\hat{q}'' -\sigma\rightarrow q''$ and $\mathcal{L}(q'') = \mathcal{L}(q')$. As σ may vary, there may be many q'' with $\mathcal{L}(q'') = \mathcal{L}(q')$. We arbitrarily choose one of them and denote it with $f(q')$. Lemma 2.1 implies that if $q'_1 \neq q'_2$, then $\mathcal{L}(q'_1) \neq \mathcal{L}(q'_2)$, yielding $f(q'_1) \neq f(q'_2)$. So $|Q''| \geq |Q'|$. If $\delta'(q', a) \neq \perp$, then some $a\rho' \in \mathcal{L}(q') = \mathcal{L}(f(q'))$, so $\delta''(f(q'), a) \neq \perp$. As a consequence, $|\delta''| \geq |\delta'|$.

If $|Q''| = |Q'|$, then f is an isomorphism. ∎

The proof has the consequence that after the end of the main loop, $Block(q_1) = Block(q_2)$ if and only if $\mathcal{L}(q_1) = \mathcal{L}(q_2)$.

Let us consider the number of times a transition (q, a, q') can be used on line 9. It is used whenever such a (B, a) is taken from $\mathcal{U}$ that $q' \in B$, that is, $Block(q') = B$. So, shortly before using (q, a, q'), $(Block(q'), a) \in \mathcal{U}$ held but ceased to hold (line 7). To use it again, $(Block(q'), a) \in \mathcal{U}$ must be made to hold again. To make $(Block(q'), a) \in \mathcal{U}$ to hold again, line 13 or 14 must be executed such that $Block(q')$ is in the role of C_{small} or C_{big}, and a is in the role of b. But line 14 tests that $(C, b) \in \mathcal{U}$, so it cannot make $(Block(q'), a) \in \mathcal{U}$ to hold if it did not hold already on line 9, although it can keep $(Block(q'), a) \in \mathcal{U}$ valid. So only line 13 can make $(Block(q'), a) \in \mathcal{U}$ to hold again. An important detail of the algorithm

is that line 13 puts the *smaller* half of C (paired with a) into $\mathcal{U}$. Therefore, each time $(Block(q'), a) \in \mathcal{U}$ starts to hold again, q' resides in a block whose size is at most half of the size in the previous time. As a consequence, (q, a, q') can be used for splitting at most $\lg |Q| + 1$ times.

3. Refinable Partitions

The refinable partition data structure maintains a partition of the set $\{1, \ldots, max\}$. Our algorithm uses one instance of it with $max = |Q|$ for the blocks and another with $max = |\delta|$ for the splitters. Each set in the partition has an index in the range $1, \ldots, sets$, where *sets* is the current number of sets. The structure supports the following operations.

Size(s)**:** Returns the number of elements in the set with index s.

Set(e)**:** Returns the index of the set that element e belongs to.

First(s) **and** *Next*(e)**:** The elements of the set s can be scanned by executing first $e := First(s)$ and then **while** $e \neq 0$ **do** $e := Next(e)$. Each element will be returned exactly once, but the ordering in which they are returned is unspecified. While scanning a set, *Mark* and *Split* must not be executed.

Mark(e)**:** Marks the element e for splitting of a set.

Split(s)**:** If either none or all elements of set s have been marked, returns 0. Otherwise removes the marked elements from the set, makes a new set of the marked elements, and returns its index. In both cases, unmarks all the elements in the set or sets.

No_marks(s)**:** Returns True if and only if none of the elements of s is marked.

The implementation uses the following max-element arrays.

*elems***:** Contains $1, \ldots, max$ in such an order that elements that belong to the same set are next to each other.

*loc***:** Tells the location of each element in *elems*, that is, $elems[loc[e]] = e$.

*sidx***:** The index of the set that e belongs to is $sidx[e]$.

first **and** *end***:** The elements of set s are $elems[f]$, $elems[f + 1]$, ..., $elems[\ell]$, where $f = first[s]$ and $\ell = end[s] - 1$.

*mid***:** Let f and ℓ be as above, and let $m = mid[s]$. The marked elements are $elems[f]$, ..., $elems[m - 1]$, and the unmarked are $elems[m]$, ..., $elems[\ell]$.

Initially $sets = 1$, $first[1] = mid[1] = 1$, $end[1] = max + 1$, and $elems[e] = loc[e] = e$ and $sidx[e] = 1$ for $e \in \{1, \ldots, max\}$. Initialization takes $O(max)$ time and $O(1)$ additional memory.

The implementation of the operations is shown in Figure 2. Each operation runs in constant time, except *Split*, whose worst-case time consumption is linear in the number M of marked elements. However, also *Split* can be treated as constant-time in the analysis of our algorithm, because it is amortized constant time. When calling *Split*, there had been M calls of *Mark*. They are unique to this call of *Split*, because *Split* unmarks the elements in question. The total time consumption of these calls of *Mark* and *Split* is $\Theta(M)$, but the same result is obtained even if *Split* is treated as constant-time.

4. Block-splitting Stage

In this section we show how lines 4...15 of the abstract algorithm can be implemented in $O(|\delta| \lg |Q|)$ time and $O(|\delta|)$ memory assuming that δ is available in a suitable ordering. The implementation of abstract lines 1...3 and 16...21 in $O(|Q| + |\delta|)$ time and memory

```
Size(s)
return end[s] − first[s]

Set(e)
return sidx[e]

First(s)
return elems[first[s]]

Next(e)
if loc[e] + 1 ≥ end[sidx[e]] then return 0
else return elems[loc[e] + 1]

Mark(e)
s := sidx[e]; ℓ := loc[e]; m := mid[s]
if ℓ ≥ m then
    elems[ℓ] := elems[m]; loc[elems[ℓ]] := ℓ
    elems[m] := e; loc[e] := m; mid[s] := m + 1

Split(s)
if mid[s] = end[s] then mid[s] := first[s]
if mid[s] = first[s] then return 0
else
    sets := sets + 1
    first[sets] := first[s]; mid[sets] := first[s]; end[sets] := mid[s]
    first[s] := mid[s]
    for ℓ := first[sets] to end[sets] − 1 do sidx[elems[ℓ]] := sets
    return sets

No_marks(s)
if mid[s] = first[s] then return True
else return False
```

Figure 2: Implementation of the refinable partition data structure

is easy and not discussed further in this paper. (By "abstract lines" we refer to lines in Figure 1).

The implementation relies on the following data structures. The "simple sets" among them are all initially empty. They have only three operations, all $O(1)$ time: the set is empty if and only if *Empty* returns True, $Add(i)$ adds number i to the set without checking if it already is there, and *Remove* removes any number from the set and returns the removed number. The implementation may choose freely the element that *Remove* removes and returns. One possible efficient implementation of a simple set consists of an array that is used as a stack.

tail, *label* **and** *head*: The transitions have the indices $1, \ldots, |\delta|$. If t is the index of the transition (q, a, q'), then $tail[t] = q$, $label[t] = a$, and $head[t] = q'$.

In_trs: This stores the indices of the input transitions of state q. The ordering of the transitions does not matter. This is easy to implement efficiently. For instance, one may use an array *elems* of size $|\delta|$, together with arrays *first* and *end* of size $|Q|$, so that the indices of the input transitions of q are $elems[first[q]]$, $elems[first[q] + 1]$,

$\ldots$, $elems[end[q]-1]$. The array can be initialized in $O(|Q|+|\delta|)$ time with counting sort, using $head[t]$ as the key.

BRP: This is a refinable partition data structure on $\{1,\ldots,|Q|\}$. It represents $\mathcal{B}$, that is, the blocks. The index of the set in *BRP* is used as the index of the block also elsewhere in the algorithm. Initially *BRP* consists of one set that contains the indices of the states.

TRP: This is a refinable partition data structure on $\{1,\ldots,|\delta|\}$. Each of the sets in it consists of the indices of the input transitions of some nonempty splitter (B,a). That is, *TRP* stores $\{\,\delta_{B,a} \mid B \in \mathcal{B} \wedge a \in \Sigma \wedge \delta_{B,a} \neq \emptyset\,\}$. The index of $\delta_{B,a}$ in *TRP* is used as the index of (B,a) also elsewhere in the algorithm. For this reason, we will occasionally use the word "splitter" also of the sets in *TRP*. Initially *TRP* consists of $\{\,\delta_{Q,a} \mid a \in \Sigma \wedge \delta_{Q,a} \neq \emptyset\,\}$, that is, two transitions are in the same set if and only if they have the same label. This can be established as follows:

```
for a ∈ Σ such that δ_{Q,a} ≠ ∅ do
    for t ∈ δ_{Q,a} do TRP.Mark(t)
    TRP.Split(1)
```

If transitions are pre-sorted such that transitions with the same label are next to each other, then this runs in $O(|\delta|)$ time and $O(1)$ additional memory.

Unready_Spls: This is a simple set of numbers in the range 1, $\ldots$, $|\delta|$. It stores the indices of the unprocessed nonempty splitters. That is, it implements the $\mathcal{U}$ of the abstract algorithm. Because each nonempty splitter has at least one incoming transition and splitters do not share transitions, $|\delta|$ suffices for the range.

Touched_Blocks: This is a simple set of numbers in the range 1, $\ldots$, $|Q|$. It contains the indices of the blocks C that were met when backwards-traversing the incoming transitions of the current splitter on abstract line 8. It is always empty on line 19.

Touched_Spls: This is a simple set of numbers in the range 1, $\ldots$, $|\delta|$. It contains the indices of the splitters that were affected when scanning the incoming transitions of the smaller of the new blocks that resulted from a split. It is empty on line 4.

The block-splitting stage is shown in Figure 3. We explain its operation in the proof of the following theorem.

Theorem 4.1. *Given a PT-DFA all whose states are relevant and that has at least one final state, the algorithm in Figure 3 computes the same $\mathcal{B}$ (represented by BRP) as lines 4...15 of Figure 1.*

Proof. Let us first investigate the operation of *Split_block*. As was told earlier, *BRP* models $\mathcal{B}$, *TRP* models the set of all nonempty splitters (or the sets of their input transitions), and *Unready_Spls* models $\mathcal{U}$. The task of *Split_block* is to update these three variables according to the splitting of a block C. Before calling *Split_block*, the states q that should go to one of the halves have been marked by calling $BRP.Mark(q)$ for each of them.

Line 1 unmarks all states of C and either splits C in *BRP* updating $\mathcal{B}$, or detects that one of the halves would be empty, so C should not be split. In the latter case, line 2 exits the procedure. The total effect of the call and its preceding calls of *BRP.Mark* is zero (except that the ordering of the states in *BRP* may have changed).

From now on assume that both halves of C are nonempty. Line 3 makes b the index of the bigger half B and b' the index of the smaller half B'. Because C is no more a block, for each $a \in \Sigma$, the pairs (C,a) are no more splitters, and must be replaced by (B,a) and (B',a), to the extent that they are nonempty. For this purpose, lines 4, 5 and 10 scan

```
    Split_block(b)
 1  b' := BRP.Split(b)
 2  if b' ≠ 0 then
 3     if BRP.Size(b) < BRP.Size(b') then b' := b
 4     q := BRP.First(b')
 5     while q ≠ 0 do
 6        for t ∈ In_trs[q] do
 7           p := TRP.Set(t)
 8           if TRP.No_marks(p) then Touched_Spls.Add(p)
 9           TRP.Mark(t)
10        q := BRP.Next(q)
11     while ¬Touched_Spls.Empty do
12        p := Touched_Spls.Remove
13        p' := TRP.Split(p)
14        if p' ≠ 0 then Unready_Spls.Add(p')
    Main_part
15  Initialize TRP to { δ_{Q,a} | a ∈ Σ ∧ δ_{Q,a} ≠ ∅ }
16  for p := 1 to TRP.sets do Unready_Spls.Add(p)
17  for q ∈ F do BRP.Mark(q)
18  Split_block(1)
19  while ¬Unready_Spls.Empty do
20     p := Unready_Spls.Remove
21     t := TRP.First(p)
22     while t ≠ 0 do
23        q := tail[t]; b' := BRP.Set(q)
24        if BRP.No_marks(b') then Touched_Blocks.Add(b')
25        BRP.Mark(q)
26        t := TRP.Next(t)
27     while ¬Touched_Blocks.Empty do
28        b := Touched_Blocks.Remove
29        Split_block(b)
```

Figure 3: Implementation of lines 4...15 of the abstract algorithm

B' and line 6 scans the incoming transitions of the currently scanned state of B'. Line 9 marks, for each $a \in \Sigma$, the transitions that correspond to (B', a). Line 7 finds the index of (C, a) in *TRP*, and line 8 adds it to *Touched_Spls*, unless it is there already. After all input transitions of B' have been scanned, lines 11 and 12 discharge the set of affected splitters (C, a). Line 13 updates (C, a) to those of (B, a) and (B', a) that are nonempty.

Line 14 corresponds to the updating of $\mathcal{U}$. If both (B, a) and (B', a) are nonempty splitters, then the index of (B', a) is added to *Unready_Spls*, that is, (B', a) is added to $\mathcal{U}$. In this case, (B, a) inherits the index of (C, a) and thus also the presence or absence in $\mathcal{U}$. If (B, a) is empty, then (B', a) inherits the index and $\mathcal{U}$-status of (C, a). If (B', a) is empty, then (C, a) does not enter *Touched_Spls* in the first place. To summarize, if $(C, a) \in \mathcal{U}$, then all of its nonempty heirs enter $\mathcal{U}$; otherwise only the smaller heir enters $\mathcal{U}$, and only if it is nonempty. This is equivalent to abstract lines 13...14. Regarding abstract line 15, (C, a) disappears automatically from $\mathcal{U}$ because its index is re-used.

Lines 15...18 implement the total effect of abstract lines 4...5. The initial value of *BRP* corresponds to $\mathcal{B} = \{Q\}$. Line 15 makes *TRP* contain the sets of input transitions of all nonempty splitters (Q, a) (where $a \in \Sigma$), and line 16 puts them all to $\mathcal{U}$. If $Q = F$, then lines 17 and 18 have no effect. Otherwise, they update $\mathcal{B}$ to $\{F, Q - F\}$, update *TRP* accordingly, and update *Unready_Spls* to contain all current nonempty splitters.

Lines 19 and 20 match trivially abstract lines 6 and 7. They choose some nonempty splitter (B, a) for processing. Lines 21...26 can be thought of as being executed between abstract lines 7 and 8. They mark the states in C_1 for every C that is scanned by abstract line 8, and collect the indices of those C into *Touched_Blocks*. Lines 27 and 28 correspond to abstract line 8, and abstract lines 9...15 are implemented by the call *Split_block*(b). Lines 1 and 2 have the same effect as abstract lines 9...11. Line 3 implements abstract line 12. The description of line 14 presented above matches abstract lines 13...15. ∎

Theorem 4.2. *Given a PT-DFA all whose states are relevant and that has at least one final state, and assuming that the transitions that have the same label are given successively in the input, the algorithm in Figure 3 runs in $O(|\delta| \lg |Q|)$ time and $O(|\delta|)$ memory.*

Proof. The data structures have been listed in this section and they all consume $O(|Q|)$ or $O(|\delta|)$ memory. Their initialization takes $O(|Q| + |\delta|)$ time. Because all states are relevant, we have $|Q| \leq |\delta| + 1$, so $O(|Q|)$ terms are also $O(|\delta|)$.

We have already seen that each individual operation in the algorithm runs in amortized constant time, except for line 15, which takes $O(|\delta|)$ time. We also saw towards the end of Section 2 that each transition is used at most $\lg |Q| + 1$ times on line 9 of the abstract algorithm. This implies that line 25, and thus lines 23...26, are executed at most $|\delta|(\lg |Q| + 1)$ times. The same holds for lines 28 and 29, because the number of *Add*-operations on *Touched_Blocks* is obviously the same as *Remove*-operations. Because *TRP*-sets are never empty, lines 20 and 21 are not executed more often than line 25, and lines 22 and 27 are executed at most twice as many times as line 25. Line 19 is executed once more than line 20, and lines 15...18 are executed once. Line 16 runs in $O(|\delta|)$ and line 17 in $O(|Q|)$ time.

Lines 1...4 are executed at most once more than line 29. If $BRP.Size(b) \geq BRP.Size(b')$ on line 3, then each of the states scanned by lines 5 and 10 was marked on line 17 or 25. Otherwise the number of scanned states is smaller than the number of marked states. Therefore, line 10 is executed at most as many times as lines 17 and 25, and line 5 at most twice as many times. Whenever lines 7...9 are executed anew (or for the first time) for some transition, the end state of the transition belongs to a block whose size is at most half of the size in the previous time (or originally), because the block was split on line 1 and the smaller half was chosen on line 3. Therefore, lines 7...9 are executed at most $|\delta| \lg |Q|$ times. Line 6 is executed as many times as lines 7 and 10 together. The executions of lines 12...14 are determined by line 8, and of line 11 by lines 4 and 8. ∎

5. Sorting Transitions

In *TRP*, transitions are sorted such that those with the same label are next to each other. Transitions are not necessarily in such an order in the input. Therefore, we must take the resources needed for sorting into account in our analysis.

Transitions can of course be sorted according to their labels with heapsort in $O(|\delta| \lg |\delta|)$ time and $O(|\delta|)$ memory. This is inferior to the time consumption of the rest of the algorithm. Because the labels need not be in alphabetical order, a suitable ordering can also

```
TRP.sets := 0
for t ∈ δ do
    a := label[t]; i := idx[a]
    if i < 1 ∨ i > TRP.sets ∨ TRP.mid[i] ≠ a then
        i := TRP.sets + 1; TRP.sets := i
        idx[a] := i; TRP.mid[i] := a; TRP.end[i] := 1
    else TRP.end[i] := TRP.end[i] + 1
TRP.first[1] := 1; TRP.end[1] := TRP.end[1] + 1; TRP.mid[1] := TRP.end[1]
for i := 2 to TRP.sets do
    TRP.first[i] := TRP.end[i − 1]
    TRP.end[i] := TRP.first[i] + TRP.end[i]; TRP.mid[i] := TRP.end[i]
for t ∈ δ do
    i := idx[label[t]]; ℓ := TRP.mid[i] − 1; TRP.mid[i] := ℓ
    TRP.elems[ℓ] := t; TRP.loc[t] := ℓ; TRP.sidx[t] := i
```

Figure 4: Initialization of TRP in $O(|\delta|)$ time and $O(|\Sigma|)$ additional memory

be found by putting the transitions into a hash table using their labels as the keys. Then nonempty hash lists are sorted and concatenated. This takes $O(|\delta|)$ time on the average, and $O(|\delta|)$ memory. However, the worst-case time consumption is still $O(|\delta| \lg |\delta|)$.

A third possibility runs in $O(|\delta|)$ time even in the worst case, but it uses $O(|\Sigma|)$ additional memory. That its time consumption may be smaller than memory consumption arises from the fact that it uses an array idx of size $|\Sigma|$ that need not be initialized at all, not even to all zeros. It is based on counting the occurrences of each label as in exercise 2.12 of [1], and then continuing like counting sort. The pseudocode is in Figure 4.

6. Measurements and Conclusions

Table 1 shows some measurements made with our test implementations of Knuutila's and our algorithm. They were written in C++ and executed on a PC with Linux and 1 gigabyte of memory. No attempt was made to optimise either implementation to the extreme. The implementation of Knuutila's algorithm completes the transition function to a full function with a well-known construction. Namely, it adds a "sink" state to which all originally absent transitions and all transitions starting from itself are directed.

The input DFAs were generated at random. Because of the difficulty of generating a precise number of transitions according to the uniform distribution, sometimes the generated number of transitions was slightly smaller than the desired number. Furthermore, the DFAs may have unreachable states and/or reachable irrelevant states that are processed separately by one or both of the algorithms. Running time depends also on the size of the minimized DFA: the smaller the result, the less splitting of blocks. We know that the joint effects of these phenomena were small, because, in all cases, the numbers of states and transitions of the minimized DFAs were $> 99.4\,\%$ of $|Q|$ and $|\delta|$ in the table. Therefore, instead of trying to avoid the imperfections by fine-tuning the input (which would be difficult), we always used the first input DFA that our generator gave for the given parameters.

The times given are the fastest and slowest of three measurements, made with $|F| = \frac{|Q|}{2} + d$, where $d \in \{-1, 0, 1\}$. They are given in seconds. The number of transitions $|\delta|$ varies between $10\,\%$ and $100\,\%$ of $|Q||\Sigma|$. The times contain the special processing of unreachable and irrelevant states, but they do not contain the reading of the input DFA from and writing

Table 1: Running time measurements. $|\delta| = p|Q||\Sigma|$, where p is given as %.
A: $|Q| = 1\,000$ and $|\Sigma| = 100$. B: $|Q| = 1\,000$ and $|\Sigma| = 1\,000$.
C: $|Q| = 10\,000$ and $|\Sigma| = 100$. D: $|Q| = 10\,000$ and $|\Sigma| = 1\,000$.

	alg.	10 %		30 %		50 %		70 %		90 %		100 %	
A	our	0.004	0.005	0.013	0.014	0.024	0.025	0.036	0.037	0.052	0.060	0.061	0.062
	Knu	0.026	0.026	0.034	0.035	0.040	0.041	0.045	0.046	0.048	0.049	0.053	0.054
B	our	0.059	0.061	0.277	0.279	0.549	0.551	0.855	0.865	1.181	1.211	1.330	1.416
	Knu	0.467	0.486	0.645	0.651	0.785	0.795	0.893	0.907	0.971	0.979	1.033	1.040
C	our	0.070	0.071	0.296	0.301	0.574	0.581	0.887	0.893	1.210	1.229	1.424	1.434
	Knu	0.526	0.529	0.730	0.734	0.901	0.904	1.027	1.035	1.128	1.130	1.200	1.202
D	our	1.224	1.238	4.038	4.087	7.132	7.164	10.50	10.57	14.18	14.34	16.41	16.48
	Knu	6.324	6.356	8.606	8.705	10.46	10.64	11.91	11.95	13.00	13.04	13.83	13.89

the result to a file. With $|Q| = |\Sigma| = 10\,000$, Knuutila's algorithm ran out of memory, while our algorithm spent about 15 s when $p = \frac{|\delta|}{|Q||\Sigma|} = 10\,\%$ and 32 s when $p = 20\,\%$.

The superiority of our algorithm when p is small is clear. That our algorithm loses when p is big may be because it uses both F and $Q - F$ in the initial splitters, whereas Knuutila's algorithm uses only one of them. Also Knuutila's algorithm speeds up as p becomes smaller. Perhaps the reason is that when p is, say, 10 %, the block that contains the sink state has an unproportioned number of input transitions, causing blocks to split to a small and big half roughly in the ratio of 10 % to 90 %. Thus small blocks are introduced quickly. As a consequence, the average size of the splitters that the algorithm uses during the execution is smaller than when $p = 100\,\%$. The same phenomenon also affects indirectly our algorithm, probably explaining why its running time is not linear in p.

Of the three notions of "smaller" mentioned in the introduction, our analysis does not apply to the other two. It seems that they would require making *Split_block* somewhat more complicated. This is a possible but probably unimportant topic for further work.

A near-future goal of us is to publish a much more complicated, true $O(m \lg n)$ algorithm for the problem in [2], that is, the multi-relational coarsest partition problem.

References

[1] Aho, A. V., Hopcroft, J. E., Ullman, J. D.: *The Design and Analysis of Computer Algorithms*. Addison-Wesley 1974.
[2] Fernandez, J.-C.: An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming* 13 (1989/90) 219–236.
[3] Gries, D.: Describing an Algorithm by Hopcroft. *Acta Informatica* 2 (1973) 97–109.
[4] Gries, D.: *The Science of Programming*. Springer 1981.
[5] Hopcroft, J.: An $n \log n$ Algorithm for Minimizing States in a Finite Automaton. *Technical Report* CS-190, Stanford University, 1970.
[6] Knuutila, T.: Re-describing an Algorithm by Hopcroft. *Theoret. Computer Science* 250 (2001) 333–363.
[7] Paige, R., Tarjan, R.: Three Partition Refinement Algorithms. *SIAM J. Computing*, 16 (1987) 973–989.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 657-668
www.stacs-conf.org

DESIGN BY MEASURE AND CONQUER A FASTER EXACT ALGORITHM FOR DOMINATING SET

JOHAN M. M. VAN ROOIJ AND HANS L. BODLAENDER

Institute of Information and Computing Sciences, Utrecht University,
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands
E-mail address: jmmrooij@cs.uu.nl
E-mail address, Hans L. Bodlaender: hansb@cs.uu.nl
URL: http://www.cs.uu.nl

Abstract. The *measure and conquer* approach has proven to be a powerful tool to *analyse* exact algorithms for combinatorial problems, like Dominating Set and Independent Set. In this paper, we propose to use measure and conquer also as a tool in the *design* of algorithms. In an iterative process, we can obtain a series of *branch and reduce* algorithms. A mathematical analysis of an algorithm in the series with measure and conquer results in a quasiconvex programming problem. The solution by computer to this problem not only gives a bound on the running time, but also can give a new reduction rule, thus giving a new, possibly faster algorithm. This makes *design by measure and conquer* a form of *computer aided algorithm design*.

When we apply the methodology to a Set Cover modelling of the Dominating Set problem, we obtain the currently fastest known exact algorithms for Dominating Set: an algorithm that uses $O(1.5134^n)$ time and polynomial space, and an algorithm that uses $O(1.5063^n)$ time.

1. Introduction

The design of fast exponential time algorithms for finding exact solutions to NP-hard problems such as Independent Set and Dominating Set has been a topic for research for over 30 years, see e.g., the results on Independent Set in the 1970s by Tarjan and Trojanowski [18, 19]. A number of different techniques have been used for these and other exponential time algorithms [5, 21, 22].

An important paradigm for the design of exact algorithms is *branch and reduce*, pioneered in 1960 by Davis and Putnam [1]. Typically, in a branch and reduce algorithm,

1998 ACM Subject Classification: F.2.2. [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—computations on discrete structures; G.2.2. [Discrete Mathematics]: Graph Theory—graph algorithms; I.2.2. [Artificial Intelligence]: Automatic Programming—automatic analysis of algorithms.

Key words and phrases: exact algorithms, exponential time algorithms, branch and reduce, measure and conquer, dominating set, computer aided algorithm design.

a collection of reduction rules and branching rules are given. Each reduction rule simplifies the instance to an equivalent, simpler instance. If no rule applies, the branching rules generate a collection of two or more instances, on which the algorithm recurses.

An important recent development in the analysis of branch and reduce algorithms is *measure and conquer*, which has been introduced by Fomin, Grandoni and Kratsch [4]. The measure and conquer approach helps to obtain good upper bounds on the running time of branch and reduce algorithms, often improving upon the currently best known bounds for exact algorithms. It has been used successfully on Dominating Set [4], Independent Set [6], Dominating Clique [15], the number of minimal dominating sets [9], Connected Dominating Set [7], Minimum Independent Dominating Set [12], and possibly others.

In this paper, we show that the measure and conquer approach can not only be used for the *analysis* of exact algorithms, but also for the *design* of such algorithms. More specifically, measure and conquer uses a non-standard size measure for instances. This measure is based on weight vectors, which are computed by solving a quasiconvex programming problem. Analysis of the solution of this quasiconvex program yields not only an upper bound to the running time of the algorithm, but also shows where we should improve the algorithm. This can lead to a new rule, which we add to the branch and reduce algorithm.

We apply this *design by measure and conquer* methodology to a Set Cover modelling of the Dominating Set problem, identical to the setting in which measure and conquer was first introduced. If we start with the trivial algorithm, then, we can obtain in a number of steps the original algorithm of Fomin et al. [4], but with additional steps, we obtain a faster algorithm, using $O(1.5134^n)$ time and polynomial space, with a variant that uses exponential memory and $O(1.5063^n)$ time. We also show that at this point we cannot improve this measure and conquer computed running time, unless we choose a different measure or change the branching rules.

While for several classic combinatorial problems, the first non-trivial exact algorithms date many years ago, for the Dominating Set problem, the first algorithms with running time $O^*(c^n)$ with $c < 2$ are from 2004, with three independent papers: by Fomin et al. [10], by Randerath and Schiermeyer [16], and by Grandoni [13]. The so far fastest algorithm for Dominating Set was found in 2005 by Fomin, Grandoni, and Kratsch [4]: this algorithm uses $O(1.5260^n)$ time and polynomial space, or $O(1.5137^n)$ time and exponential space.

2. Preliminaries

Given a collection of non-empty sets $\mathcal{S}$, a *set cover* of $\mathcal{S}$ is a subset $\mathcal{C} \subseteq \mathcal{S}$ such that every element in any of the sets in $\mathcal{S}$ occurs in some set in $\mathcal{C}$. In the Set Cover problem we are given a collection $\mathcal{S}$ and are asked to compute a set cover of minimum cardinality.

The universe $\mathcal{U}_\mathcal{S}$ of a Set Cover problem instance is the set of all elements in any set in $\mathcal{S}$; $\mathcal{U}_\mathcal{S} = \bigcup_{S \in \mathcal{S}} S$. The frequency $f(e)$ of an element $e \in \mathcal{U}_\mathcal{S}$ is the number of sets in $\mathcal{S}$ in which this element occurs. Let $\mathcal{S}(e) = \{S \in \mathcal{S} \mid e \in S\}$ be the set of sets in $\mathcal{S}$ in which the element e occurs. We define a connected component $\mathcal{C}$ in a Set Cover problem instance $\mathcal{S}$ in a natural way: a minimal non-empty subset $\mathcal{C} \subseteq \mathcal{S}$ for which all elements in the sets in $\mathcal{C}$ occur only in sets contained in $\mathcal{C}$. The dimension $d_\mathcal{S}$ of a Set Cover problem instance is the sum of the number of sets in $\mathcal{S}$ and the number of elements in $\mathcal{U}_\mathcal{S}$; $d_\mathcal{S} = |\mathcal{S}| + |\mathcal{U}_\mathcal{S}|$.

Let $G = (V, E)$ be an undirected graph. A subset $D \subseteq V$ of nodes is called a *dominating set* if every node $v \in V$ is either in D or adjacent to some node in D. The Dominating Set problem is to compute for a given graph G a dominating set of minimum cardinality.

We can reduce the minimum dominating set problem to the SET COVER problem by introducing a set for each node of G which contains the node itself and its neighbours; $\mathcal{S} := \{N[v] \mid v \in V\}$. Thus we can solve a DOMINATING SET problem on a graph of n nodes by a minimum set cover algorithm running on an instance of dimension $d = 2n$.

Both problems are long known to be NP-complete [11, 14], which motivates the search for fast exponential time algorithms.

3. A Faster Exact Algorithm for Dominating Set

In this section, we give our new algorithm for DOMINATING SET. The algorithm is an improvement to the algorithm by Fomin et al. [4]; it is obtained from this algorithm by adding some additional reduction rules. These rules were derived using the design by measure and conquer approach, see Section 4. After introducing our algorithm, we recall the necessary background of the measure and conquer method [4].

3.1. The Algorithm

Our algorithm for the DOMINATING SET problem uses the SET COVER modelling of DOMINATING SET shown in Section 2. It is a branch and reduce algorithm on this modelling consisting of four simple reduction rules, one base case for the recursion and a branching rule. See Algorithm 1.

For a given problem instance we first apply the following reduction rules:

(1) *Base case.* If all sets in the instance are of size at most two then finding a minimum set cover is equivalent to finding a maximum matching in a graph. Introduce a node for each element and an edge for each set of size two. Now the maximum matching joined with some sets containing the unmatched nodes form a minimum set cover. This matching can be computed in polynomial time [2].

(2) *Splitting connected components.* If the instance contains multiple connected components, compute the minimum set cover in each connected component separately.

(3) *Subset rule.* If the instance contains sets S_1, S_2 with $S_1 \subseteq S_2$, then we remove S_1 from the instance. Namely, in each minimum set cover that contains S_1, we can replace S_1 by S_2 and obtain a minimum set cover without S_2.

(4) *Subsumption rule.* If the set of sets $S(e')$ in which an element e' occurs is a subset of the set of sets $S(e)$ in which another element e occurs, we remove the element e. For any set cover, covering e' also covers e.

(5) *Unique element or singleton rule.* If any set of size one remains in the instance after application of the previous rules, we add this set to the set cover. For the element in this set must occur uniquely in this set, otherwise it would have been a subset of another set and have been removed by rule 3.

(6) *Avoiding unnecessary branchings based on frequency two elements.* For any set S in the problem instance let r_2 be the number of frequency two elements it contains. Let m be the number of elements in the union of sets containing the other occurrences of these frequency two elements, excluding any element already in S. If for any set S: $m < r_2$ then include S in the set cover.
This rule is correct since if we would branch on S and include it in the set cover we would cover $|S|$ elements with one set. If the set cover does not use S, it must use all sets containing the other occurrence of the frequency two elements, since they have

become unique elements now. Notice that by Rule 4 all other occurrences of the frequency two elements must be in different sets and thus we would cover $|S| + m$ elements with r_2 sets. So if $m < r_2$ the first case can be preferred over the second: we can just add S to the cover and have $r_2 - 1 \leq m$ sets left to cover at least this much elements.

For the branching rule, we select a set of maximum cardinality and create two subproblems by either including it in the minimum set cover and removing all newly covered elements from the problem instance or removing it.

Algorithm 1 Algorithm Designed by Measure and Conquer

$\text{MSC}(\mathcal{S}) = \{$
if $\max\{|S| \mid S \in \mathcal{S}\} \leq 2$ **then**
  return minimum set cover of $\mathcal{S}$ by computing a matching
else if $\exists \mathcal{C} \subseteq \mathcal{S} : \mathcal{C} \notin \{\emptyset, \mathcal{S}\}, \{S(e) | e \in S, S \in \mathcal{C}\} = \mathcal{C}$ **then**
  return $\text{MSC}(\mathcal{C}) + \text{MSC}(\mathcal{S} \backslash \mathcal{C})$
else if $\exists S, S' \in \mathcal{S} : S \neq S',\ S \subseteq S'$ **then**
  return $\text{MSC}(\mathcal{S} \backslash \{S\})$
else if $\exists e, e' \in \mathcal{U}_\mathcal{S} : e \neq e',\ \mathcal{S}(e) \supseteq \mathcal{S}(e')$ **then**
  $\mathcal{S}' = \text{MSC}(\{S \backslash \{e\} | S \in \mathcal{S}\})$
  return $\{S \cup \{e\} | S \in \mathcal{S}', S \cup \{e\} \in \mathcal{S}\} \cup \{S | S \in \mathcal{S}', S \cup \{e\} \notin \mathcal{S}\}$
else if $\exists \{e\} \in \mathcal{S}$ **then**
  return $\{\{e\}\} \cup \text{MSC}(\mathcal{S} \backslash \{\{e\}\})$
else if $\exists S : |\bigcup \{S' \backslash S | e \in S, f(e) = 2, S' \in \mathcal{S}(e)\}| < |\{e \in S | f(e) = 2\}|$ **then**
  return $S \cup \text{MSC}(\{S' \backslash S | S' \in \mathcal{S} \backslash \{S\}\})$
else
  Let $S := \text{argmax}_{S' \in \mathcal{S}}(|S'|)$
  $\mathcal{P} = \{\text{MSC}(\mathcal{S} \backslash \{S\}), S \cup \text{MSC}(\{S' \backslash S | S' \in \mathcal{S} \backslash \{S\}\})\}$
  return $\text{argmin}_{P \in \mathcal{P}}(|P|)$
end if
$\}$

3.2. Running time analysis by Measure and Conquer

The basic idea of *measure and conquer* is the usage of a non-standard measure for the complexity of a problem instance in combination with an extensive subcase analysis. In the case of Set Cover, we give weights to set sizes and element frequencies, and sum these weights over all items and sets. We enumerate many subcases in which the algorithm can branch and derive recurrence relations for each of these cases in terms of these weights. Finally we obtain a large numerical optimisation problem which computes the weights corresponding to the smallest solution to all recurrence relations, giving an upper bound on the running time of our algorithm. This analysis is similar to [4].

We let $v_i, w_i \in [0, 1]$ be the weight of an element of frequency i and a set of size i respectively, and set our *variable measure of complexity* $k_\mathcal{S}$ to:

$$k_\mathcal{S} = \sum_{S \in \mathcal{S}} w_{|S|} + \sum_{e \in \mathcal{U}_\mathcal{S}} v_{f(e)} \qquad \text{notice}: \quad k_\mathcal{S} \leq d_\mathcal{S}$$

Sets of different sizes and elements of different frequencies contribute equally to the dimension of the instance, but now larger sets and higher frequency elements can contribute more to the measured complexity of the instance. Furthermore we set $v_i = w_i = 0, i \in \{0, 1\}$ since all frequency one elements and size one sets are removed by the reduction rules. For later use we introduce quantities representing the reduction in problem complexity when the size of a set or the frequency of an element is reduced by one. For technical reasons, these quantities must be non-negative.

$$\Delta w_i = w_i - w_{i-1} \qquad \Delta v_i = v_i - v_{i-1} \qquad \forall i \geq 1 : \Delta v_i, \Delta w_i \geq 0$$

The next step will be to derive recurrence relations representing problem instances the algorithm branches on. Let $N(k)$ be the number of subproblems generated in order to solve a problem of measured complexity k. And let Δk_{in} (include S) and Δk_{out} (discard S) be the difference in measured complexity of both subproblems compared to the problem instance we branch on. Finally let $|S| = \sum_{i=2}^{\infty} r_i$, where r_i the number of elements in S of frequency i.

If we add S to the set cover, S is removed together with all its elements. This results in a reduction in size of $w_{|S|} + \sum_{i=2}^{\infty} r_i v_i$. Because of the removal of these elements, other sets are reduced in size; this leads to an additional complexity reduction of at least $\min_{j \leq |S|}\{\Delta w_j\} \sum_{i=2}^{\infty}(i-1)r_i$. To keep the formula (and the next) linear, we set $\min_{j \leq |S|}\{\Delta w_j\} = \Delta w_{|S|}$ and keep the formula correctly modelling the algorithm by introducing the following constraints on the weights:

$$\forall i \geq 2 : \Delta w_i \leq \Delta w_{i-1}$$

One can show that including these in the numerical optimisation problem does not change the solution, as it gives the same weights. The constraints help to considerably speed up this optimisation process.

If we discard S, we also remove it from the problem instance, and hence all its elements are reduced in frequency by one. So we have a complexity reduction of $w_{|S|} + \sum_{i=2}^{\infty} r_i \Delta v_i$. Besides this reduction, the sets which contain the second occurrences of any frequency two element are included in the set cover. Notice that these must be different sets due to reduction Rule 4. Because of Rule 6 we know that at least r_2 other elements must be in these sets as well, and these also must occur somewhere else in the instance, hence even more sets are reduced in size. Summation leads to an additional size reduction of $r_2(v_2 + w_2 + \Delta w_{|S|})$. Here we also use Rule 2 to make sure that not all these frequency two elements occur in the same set, because in that case all considered sets form a connected component of at most five sets which thus can be solved in $O(1)$ time.

This leads to the following set of recurrence relations: $\forall\, 3 \leq |S| = \sum_{i=2}^{\infty} r_i$:

$$N(k) \leq N(k - \Delta k_{out}) + N(k - \Delta k_{in})$$

where

$$\begin{aligned} \Delta k_{out} &= w_{|S|} + \sum_{i=2}^{\infty} r_i \Delta v_i + r_2(v_2 + w_2 + \Delta w_{|S|}) \\ \Delta k_{in} &= w_{|S|} + \sum_{i=2}^{\infty} r_i v_i + \Delta w_{|S|} \sum_{i=2}^{\infty}(i-1)r_i \end{aligned}$$

We make the problem finite by setting for some large enough p all $v_i = w_i = 1$ for $i \geq p$, and only consider the subcases $|S| = \sum_{i=2}^{p} r_i + r_{>p}$, where $r_{>p} = \sum_{i=p+1}^{\infty} r_i$. Now

we have a finite set of recurrences which model our algorithm since the recurrences for the cases where $|S| > p + 1$ are dominated by those where $|S| = p + 1$. The best value for p follows from the optimisation, for if chosen too small the now constant recurrences (weights equal 1) will dominate all others in the optimum, and if chosen too large the extra v_i and w_i are optimised to 1 (and the optimisation problem was unnecessarily hard). Here p equals 7.

A solution to this set of recurrence relations will be of the form $N(k) = \alpha^k$, where α is the smallest solution of the set of inequalities:

$$\alpha^k \leq \alpha^{k-\Delta k_{out}} + \alpha^{k-\Delta k_{in}}$$

Since $k \leq d$ where d the dimension of the problem, we know that the algorithm will have a running time of $O((\alpha + \epsilon)^d)$, for any $\epsilon > 0$:

$$O(poly(d)N(k)) = O(poly(d)\alpha^k) \leq O(poly(d)\alpha^d) \leq O((\alpha + \epsilon)^d)$$

From here on we let ϵ be the error in the upward decimal rounding of α.

So for any given vector $\vec{v} = (0, v_2, v_3, v_4, \ldots)$ and $\vec{w} = (0, w_2, w_3, w_4, \ldots)$ we can now compute the running time measured with these weights. As a result we have obtained a numerical optimisation problem: choose the best weights so that the upper bound on the running time is minimal.

The numerical solution to this problem can be found in the last cell of Table 1, resulting in an upper bound on the running time of the algorithm of $O(1.2302^d)$:

3.3. Quasiconvex programming

The sort of numerical optimisation problems arising from measure and conquer analyses are *quasiconvex programs*, named after the kind of function we are optimising: a *quasiconvex function*, which is a function with convex level sets $\{\vec{x} \mid q(x) \leq \lambda\}$.

To our knowledge there are currently two different techniques in use to solve these quasiconvex programs: randomised search, and Eppstein's *smooth quasiconvex programming* algorithm [3]. We have implemented a variant of the second technique; for details see [20].

3.4. Results

As discussed, we have now obtained the following result.

Theorem 3.1. *Algorithm 1 solves a* Set Cover *problem instance of dimension d in $O(1.2302^d)$ time and polynomial space.* ∎

Using the minimum set cover modelling of Dominating Set this results in:

Corollary 3.2. *There exists an algorithm that solves the* Dominating Set *problem in $O(1.5134^n)$ time and polynomial space.* ∎

We can further reduce the time complexity of the algorithm at the cost of exponential space. This can be done by dynamic programming; the algorithm keeps track of all solutions to subproblems solved and if the same subproblem turns up more than ones it is looked up. Notice that querying and storing the subproblems can be implemented in polynomial time.

We compute the new time complexity based on [8, 17] and obtain:

Theorem 3.3. *Algorithm 1, modified as above, solves a* Set Cover *problem of dimension d in $O(1.2273^d)$ time and space.* ∎

Corollary 3.4. *There exists an algorithm that solves the* Dominating Set *problem in* $O(1.5063^n)$ *time and space.* ∎

4. Design by Measure and Conquer

The beauty of our algorithm lies in the fact that it has been designed using a form of *computer aided algorithm design* which we call *design by measure and conquer*. Given a variable measure of complexity as in the analysis in Section 3.2 and a set of branching rules, all polynomial time computable reduction rules relative to this measure and branching rules follow by the method. We start with a trivial branch and reduce algorithm, i.e. one without any reduction rules and only consisting of the branching rule and a trivial base case (if the problem is empty, return $\emptyset$). Next we exhaustively apply an improvement step, which comes up with a new reduction rule and hence a possibly faster algorithm. This changes the algorithm analysis technique measure and conquer into a technique for algorithm design.

Thus, this gives a very nice process, where a human invents additional reduction rules, and the computational power of our computer does the extensive measure and conquer analysis and points to all possible points of direct improvement. This combination has proven to be successful as we see from the results of Section 3. While constructing our algorithm, the previously fastest algorithm for Dominating Set by Fomin et al. [4] has been obtained as an intermediate step. It has now been improved up to a point where we need to either change the branching rule (or add new branching rules) or modify the measure and conquer analysis, i.e. use a different variable measure or perform a more elaborate subcase analysis. See Table 1 for information on the analysis and added rules for all algorithms, from the starting trivial algorithm without any reduction rules till we obtain Algorithm 1.

4.1. A Single Iteration: improving the previously fastest algorithm

We now demonstrate how the improvement step works, by giving one such improvement as elaborate example, namely an improvement we can make when we start with the algorithm by Fomin et al. [4]. This step is marked with a star in Table 1. First we perform a measure and conquer analysis on the current algorithm giving us the optimal instantiation of the variable measure, and an upper bound on the running time of the algorithm. Next we examine the quasiconvex function we have just optimised.

The quasiconvex function has the following form:

$$q(\vec{v},\vec{w}) = \max_{c\in\mathcal{C}} q_c(\vec{v},\vec{w}) = \max_{c\in\mathcal{C}} \left\{ \alpha \in \mathbb{R}_{>0} \,|1 = \alpha^{-\Delta k^c_{out}} + \alpha^{-\Delta k^c_{in}} \right\}$$

where $\mathcal{C}$ is the set of all possible instances the algorithm can branch on and $\Delta k^c_{in}, \Delta k^c_{out}$ are the differences in measured complexity between the generated subproblems and branching subcase c.

Each one of the functions q_c is quasiconvex (see [3]), i.e. it has convex level sets. The situation is very similar to finding the point x of minimum maximum distance to a set of points P in N dimensional space: only a few points in P have distance to x tight to this maximum, and moving away from x always results in at least one of these distances to increase. If one such tight point is moved or removed, this directly influences the optimum x and the minimum maximum distance.

We now consider the eight cases that are tight to the value of the quasiconvex function q in the optimum. These are:

$$|S| = r_2 = 3, \quad |S| = r_3 = 3, \quad |S| = r_4 = 3, \quad |S| = r_5 = 4$$
$$|S| = r_6 = 4, \quad |S| = r_6 = 5, \quad |S| = r_7 = 5, \quad |S| = r_7 = 6$$

Now, if we can formulate a reduction rule that either further reduces the size of any subproblem generated in these cases, or removes any of these cases completely, then we lower the value of the corresponding q_c, or remove this q_c respectively, resulting in a new optimum corresponding to a faster running time.

We take the simplest case for improvement; $|S|$ as small as possible, and with as low frequency elements as possible. This corresponds to an instance with:

$$S = \{ 1, 2, 3 \} \quad \text{existing next to:} \quad \{ 1, 2, 4 \}, \{ 3, 4 \}$$

We emphasize that this is not an entire instance, but just a fragment of an instance containing the set S used for branching and the collection of sets in which the elements from S also occur. In an instance corresponding to this subcase the element 4 can be of frequency two or higher, but all sets are of size three or smaller.

We note that we do not need to branch on this particular subcase: elements 1 and 2 occur in exactly the same sets, and thus if a set cover covers one of these, the other is covered as well. We generalise this and formulate the subsumption rule (Rule 4 of Algorithm 1). Now we have a new algorithm, for which we can adjust the measure and conquer analysis, and repeat this process.

4.2. The Process Halts

Above, we discussed how to perform one step of the design by measure and conquer process. For a complete overview of the construction of Algorithm 1 see Table 1, with the relevant data for each improvement step. Note from Table 1 that after each new step, the example worst case instance part is no longer a valid worst case for the next step. As a result, at each step either some subcases are removed by using a larger smallest set S or by removing small sets or elements (setting $v_1 = 0$ or $w_1 = 0$), or the size reduction in the formula for Δk_{out} is increased. After each step we refactored the reduction rules and removed possible superfluous ones. We have not included the formula for Δk_{in} in this table, since it does not change except that $r_1 \neq 0$ in early stages.

It appears that we must use a different approach to obtain a faster algorithm. Considered the following problem:

Problem 4.1. Given a Set Cover instance $\mathcal{S}$ and a set $S \in \mathcal{S}$ with the properties:

(1) Non of the reduction rules of Algorithm 1 apply to $\mathcal{S}$.
(2) All sets in $\mathcal{S}$ have cardinality at most three; $|S| = 3$.
(3) Every element $e \in S$ has frequency two.

Question: Does there exist a minimum set cover of $\mathcal{S}$ containing S?

Proposition 4.2. *Problem 4.1 is NP-complete.*

Proposition 4.2 implies that we cannot formulate a polynomial time reduction rule that removes the current simplest worst case of our algorithm by deciding on whether S is in a minimum set cover or not, unless $P = NP$.

Latest new reduction rule	Running times for SET COVER and DOMINATING SET
current formula for Δk_{out}	
subcases considered	instance part of the simplest worst case;
weights vectors $\vec{v}$ and $\vec{w}$	S − other occurrences of elements of S

Trivial algorithm	$O(1.4519^d)$ $O(2.1080^n)$
$w_{\|S\|} + \sum_{i=1}^{\infty} r_i \Delta v_i$	
$1 \leq \|S\| = \sum_{i=1}^{p} r_i + r_{>p} \leq p + 1 = 3$	$\{ 1 \}$ − $\emptyset$
$\vec{v} = (0.8808, 0.9901, \ldots)$	$\vec{w} = (0.9782, \ldots)$
Stop when all sets of size one	$O(1.3380^d)$ $O(1.7902^n)$
$w_{\|S\|} + \sum_{i=1}^{\infty} r_i \Delta v_i$	
$2 \leq \|S\| = \sum_{i=1}^{p} r_i + r_{>p} \leq p + 1 = 4$	$\{ 1, 2 \}$ − $\emptyset$
$\vec{v} = (0.7289, 0.9638, 0.9964, \ldots)$	$\vec{w} = (0.4615, 0.9229, \ldots)$
Include all frequency one elements	$O(1.2978^d)$ $O(1.6842^n)$
$w_{\|S\|} + \sum_{i=2}^{\infty} r_i \Delta v_i + \delta_{r_2>0} w_1 + \delta_{\|S\|=r_2=2} \Delta w_2$	
$2 \leq \|S\| = \sum_{i=2}^{p} r_i + r_{>p} \leq p + 1 = 5$	$\{ 1, 2 \}$ − $\{ 1, 2 \}$
$\vec{v} = (0, 0.4818, 0.8357, 0.9636, \ldots)$	$\vec{w} = (0.4240, 0.8480, 0.9676, \ldots)$
Subset rule	$O(1.2665^d)$ $O(1.6038^n)$
$w_{\|S\|} + \sum_{i=2}^{\infty} r_i \Delta v_i + \delta_{r_2>0}(w_2 + v_2) + \delta_{\|S\|=r_2=2} \Delta w_2$	
$2 \leq \|S\| = \sum_{i=2}^{p} r_i + r_{>p} \leq p + 1 = 7$	$\{ 1, 2 \}$ − $\{ 1, 3 \}, \{ 1, 4 \}, \{ 2, 3 \}, \{ 2, 4 \}$
$\vec{v} = (0, 0.3900, 0.7992, 0.9318, 0.9808, \ldots)$	$\vec{w} = (0, 0.6973, 0.9093, 0.9800, \ldots)$
Compute matching for size two sets*	$O(1.2352^d)$ $O(1.5258^n)$
$w_{\|S\|} + \sum_{i=2}^{\infty} r_i \Delta v_i + \delta_{r_2>0}(w_2 + v_2) + \delta_{\|S\|=3, r_2 \geq 2}(\Delta w_3 + \delta_{r_2=3} w_2) + \delta_{\|S\|=r_2=4} w_4$	
$3 \leq \|S\| = \sum_{i=2}^{p} r_i + r_{>p} \leq p + 1 = 7$	$\{ 1, 2, 3 \}$ − $\{ 1, 2, 4 \}, \{ 3, 4 \}$
$\vec{v} = (0, 0.3978, 0.7650, 0.9263, 0.9842, \ldots)$	$\vec{w} = (0, 0.3787, 0.7575, 0.9103, 0.9763, \ldots)$
Subsumption rule	$O(1.2339^d)$ $O(1.5223^n)$
$w_{\|S\|} + \sum_{i=2}^{\infty} r_i \Delta v_i + \delta_{r_2>0}(r_2 w_2 + v_2) + \delta_{\|S\|=r_2=3} \Delta v_3$	
$3 \leq \|S\| = \sum_{i=2}^{p} r_i + r_{>p} \leq p + 1 = 7$	$\{ 1, 2, 3 \}$ − $\{ 1, 4 \}, \{ 2, 4 \}, \{ 3, 4 \}$
$\vec{v} = (0, 0.3545, 0.7455, 0.9203, 0.9818, \ldots)$	$\vec{w} = (0, 0.3755, 0.7510, 0.9061, 0.9745, \ldots)$
Avoid unnecessary branchings	$O(1.2313^d)$ $O(1.5160^n)$
$w_{\|S\|} + \sum_{i=2}^{\infty} r_i \Delta v_i + r_2(w_2 + v_2) + \delta_{r_2>1}(r_2 - 1) \Delta w_{\|S\|}$	
$3 \leq \|S\| = \sum_{i=2}^{p} r_i + r_{>p} \leq p + 1 = 8$	$\{ 1, 2, 3 \}$ − $\{ 1, 4 \}, \{ 2, 5 \}, \{ 3, 6 \}$
$\vec{v} = (0, 0.269912, 0.689810, 0.892666, 0.965849, 0.992140, \ldots)$	
$\vec{w} = (0, 0.376088, 0.752176, 0.907558, 0.974394, 0.999212, \ldots)$	
Connected components (final)	$O(1.2302^d)$ $O(1.5134^n)$
$w_{\|S\|} + \sum_{i=2}^{\infty} r_i \Delta v_i + r_2(w_2 + v_2 + \Delta w_{\|S\|})$	
$3 \leq \|S\| = \sum_{i=2}^{p} r_i + r_{>p} \leq p + 1 = 8$	$\{ 1, 2, 3 \}$ − $\{ 1, 4 \}, \{ 2, 5 \}, \{ 3, 6 \}$
$\vec{v} = (0, 0.219478, 0.671386, 0.876555, 0.956850, 0.988195, \ldots)$	
$\vec{w} = (0, 0.375418, 0.750835, 0.905768, 0.971965, 0.998158, \ldots)$	

* Algorithm by Fomin, Grandoni and Kratsch [4].

Table 1: The iterations of the design by measure and conquer process.

We can construct similar NP-complete problems for all other worst cases of Algorithm 1. Therefore Algorithm 1 is optimal in some sense: we cannot straightforwardly improve it by performing another iteration. In order to obtain a faster branch and reduce algorithm using polynomial time reduction rules with a smaller measure and conquer proved time bound, it is necessary to either change the variable measure, the branching rule(s), or perform a more extensive subcase analysis.

Very recently, we pursued the last option with little result. We tried to further subdivide the frequency two elements in the branch set depending on the size of the set containing their second occurence (two or larger) and if this is a set of size two, on the frequency of the other element in this set. This resulted in a set of very technical reduction rules and a small speedup for the case where we use only polynomial space. This speedup, however, was almost completely lost when using exponential space because some of the weights involved became almost zero.

5. Conclusion and Further Research

In this paper, we have given the currently fastest exact algorithm for the Dominating Set problem. Besides setting the current record for this central graph theoretic problem, we also have shown that measure and conquer can be used as a tool for the design of algorithms.

We have shown that there exists a strong relation between the chosen variable measure, the branching rule(s) and the reduction rules of a measure and conquer based algorithm. We intend to further investigate this relation and examine to what point we can deduce not only reduction rules, but also branching rules from the given measure.

We plan to apply the *design by measure and conquer* method to a number of other combinatorial problems, and hope and expect that in a number of cases, such a computer aided algorithm design will give further improvements to the best known exact algorithms for these problems.

In this paper, we observe that measure and conquer can be used as a form of *computer aided algorithm design*. Another intriguing question is whether we can automate some additional steps in the design process, e.g., can we automatically obtain reduction rules from the solution of the quasiconvex program?

References

[1] M. David and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7:201–215, 1960.
[2] J. Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17:445–467, 1965.
[3] D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004*, pages 781–790, 2004.
[4] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: Domination — a case study. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, ICALP 2005*, pages 191–203. Springer Verlag, Lecture Notes in Computer Science, vol. 3580, 2005.
[5] F. V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87:47–77, 2005.
[6] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 18–25, 2006.

[7] F. V. Fomin, F. Grandoni, and D. Kratsch. Solving connected dominating set faster than 2^n. In S. Arun-Kumar and N. Garg, editors, *Proceedings 26th International Comference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2006*, pages 152–163. Springer Verlag, Lecture Notes in Computer Science, vol. 4337, 2006.

[8] F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. Technical Report 359, Department of Informatics, University of Bergen, Bergen, Norway, 2007.

[9] F. V. Fomin, F. Grandoni, A. Pyatkin, and A. A. Stepanov. Bounding the number of minimal dominating sets: a measure and conquer approach. In *Proceedings 16th International Symposium on Algorithms and Computation, ISAAC 2005*, pages 192–203. Springer Verlag, Lecture Notes in Computer Science, vol. 3827, 2005.

[10] F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In J. Hromkovič, M. Nagl, and B. Westfechtel, editors, *Proceedings 30th International Workshop on Graph-Theoretic Concepts in Computer Science WG'04*, page 245=256. Springer Verlag, Lecture Notes in Computer Science, vol. 3353, 2004.

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

[12] S. Gaspers and M. Liedloff. A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. In F. V. Fomin, editor, *Proceedings 32nd International Workshop on Graph-Theoretic Concepts in Computer Science WG'06*, pages 78–89. Springer Verlag, Lecture Notes in Computer Science, vol. 4271, 2006.

[13] F. Grandoni. A note on the complexity of minimum dominating set. *J. Disc. Alg.*, 4:209–214, 2006.

[14] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85 – 104. Plenum Press, 1972.

[15] D. Kratsch and M. Liedloff. An exact algorithm for the minimum dominating clique problem. In H. L. Bodlaender and M. A. Langston, editors, *Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006*, pages 128–139. Springer Verlag, Lecture Notes in Computer Science, vol. 4169, 2006.

[16] B. Randerath and I. Schiermeyer. Exact algorithms for minimum dominating set. Technical Report zaik2005-501, Universität zu Köln, Cologne, Germany, 2005.

[17] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7:425–440, 1986.

[18] R. E. Tarjan. Finding a maximum clique. Technical report, Department of Computer Science, Cornell University, Ithaca, NY, 1972.

[19] R. E. Tarjan and A. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6:537–546, 1977.

[20] J. M. M. van Rooij. Design by measure and conquer: An $O(1.5086^n)$ algorithm for minimum dominating set and similar problems. Master's thesis, Institute for Information and Computing Sciences, Utrecht University, 2006.

[21] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization: "Eureka, you shrink"*, pages 185–207, Berlin, 2003. Springer Lecture Notes in Computer Science, vol. 2570.

[22] G. J. Woeginger. Space and time complexity of exact algorithms: some open problems (invited talk). In R. G. Downey and M. R. Fellows, editors, *Proceedings 1st International Workshop on Parameterized and Exact Computation, IWPEC 2004*, pages 281–290, Berlin, 2004. Springer Lecture Notes in Computer Science, vol. 3162.

Symposium on Theoretical Aspects of Computer Science 2008 (Bordeaux), pp. 669-680
www.stacs-conf.org

WEIGHTED MATCHING IN THE SEMI-STREAMING MODEL

MARIANO ZELKE

Humboldt-Universität zu Berlin, Institut für Informatik, 10099 Berlin
E-mail address: zelke@informatik.hu-berlin.de
URL: http://www2.informatik.hu-berlin.de/∼zelke/

Abstract. We reduce the best known approximation ratio for finding a weighted matching of a graph using a one-pass semi-streaming algorithm from 5.828 to 5.585. The semi-streaming model forbids random access to the input and restricts the memory to $\mathcal{O}(n \cdot \operatorname{polylog} n)$ bits. It was introduced by Muthukrishnan in 2003 and is appropriate when dealing with massive graphs.

1. Introduction

Matching. Consider an undirected graph $G = (V, E)$ without multi-edges or loops, where n and m are the number of vertices and edges, respectively. Let furthermore $w : E \to \mathbb{R}^+$ be a function that assigns a positive weight $w(e)$ to each edge e. A *matching* in G is a subset M of the edges such that no two edges in M have a vertex in common. With $w(M) := \sum_{e \in M} w(e)$ being the weight of M, the *maximum weighted matching problem* MWM is to find a matching in G that has maximum weight over all matchings in G.

That problem is well studied and exact solutions in polynomial time are known, see [12] for an overview. The fastest algorithm is due to Gabow[4] and runs in time $\mathcal{O}(nm+n^2 \log n)$.

Approximation Algorithms. When processing massive graphs even the fastest exact algorithms computing an MWM are too time-consuming. Examples where weighted matchings in massive graphs must be calculated are the refinement of FEM nets [7] and multilevel partitioning of graphs [8].

To deal with such graphs there has been effort to find algorithms that in a much shorter running time compute solutions that are not necessarily optimal but have some guaranteed quality. Such algorithms are called *approximation algorithms* and their performance is given by an *approximation ratio*. A matching algorithm achieves a c-approximation ratio if for every graph G the algorithm finds a matching M in G such that $w(M) \geq \frac{w(M^*)}{c}$, where M^* is a matching of maximum weight in G.

A 2-approximation algorithm computing a matching in time $\mathcal{O}(m)$ was given by Preis [11]. The best known approximation ratio approachable in linear time is $(3/2 + \varepsilon)$ for an arbitrarily small but constant ε. This ratio is obtained by an algorithm of Drake and

1998 ACM Subject Classification: G.2.2, F.2.2.
Key words and phrases: semi-streaming algorithm, matching, approximation algorithm, graph algorithm.
Supported by the DFG Research Center Matheon "Mathematics for key technologies" in Berlin.

Hougardy[1] in time $\mathcal{O}(m \cdot \frac{1}{\varepsilon})$, an algorithm of Pettie and Sanders[10] gets the same ratio slightly faster in time $\mathcal{O}(m \cdot \log \frac{1}{\varepsilon})$.

Streaming Model. The large amount of input for today's computational tasks often exceeds the size of the working memory and can only be stored on disks or even tapes in total. The key assumption of the traditional RAM model, that is, a working memory containing the whole input allowing very fast random access to every input item, is therefore put in question. Rather seek times of read/write heads are dominating the running time. Thus for algorithms as the above ones that do not consider the peculiarities of external memory the running time totally gets out of hand.

To develop time-efficient algorithms working on these storage devices it is reasonable to assume the input of the algorithm (which is the output of the storage devices) to be a sequential stream. While tapes produce a stream as their natural output, disks reach much higher output rates when presenting their data sequentially in the order it is stored.

Streaming algorithms are developed to deal with such large amounts of data arriving as a stream. In the classical *data stream model*, see e.g. [5], [9], the algorithm has to process the input stream using a working memory that is small compared to the length of the input. In particular the algorithm is unable to store the whole input and therefore has to make space-efficient summarizations of it according to the query to be answered.

Semi-Streaming Model. With the objective of approaching graph problems in the streaming context Muthukrishnan[9] proposed the model of a *semi-streaming algorithm*: Random access to the input graph G is forbidden, on the contrary the algorithm gets the edges of G in arbitrary order as the input stream. The memory of the algorithm is restricted to $\mathcal{O}(n \cdot \operatorname{polylog} n)$ bits. That does not suffice to store all edges of G if G is sufficiently dense, i.e., $m = \omega(n \cdot \operatorname{polylog} n)$. A semi-streaming algorithm may read the input stream for a number of P passes. The parameter T denotes the *per-edge processing time*, that is, the time the algorithm needs to handle a single edge.

Despite the heavy restrictions of the model there has been progress in developing semi-streaming algorithms solving graph problems. Feigenbaum et al.[2], [3] presented semi-streaming algorithms for testing k-vertex and k-edge connectivity of a graph, k being a constant. They pointed out how to find the connected components and a bipartition and how to calculate a minimum spanning tree of a weighted graph. Zelke[13] showed how all these problems can be solved using only a constant per-edge processing time.

Matching in the Semi-Streaming Model. There are approaches to find a weighted matching of a graph in the semi-streaming model. McGregor[6] presents an algorithm finding a $(2+\varepsilon)$-approximative solution with a number of passes $P > 1$ depending on ε.

However, for some real-world applications even a second pass over the input stream is unfeasible. If observed phenomena are not stored and must be processed immediately as they happen only a single pass over the input can occur. For the case of one-pass semi-streaming algorithms it is known, see [2], that finding the optimal solution to the MWM problem is impossible in general graphs. A first one-pass semi-streaming algorithm approximating the MWM problem with a ratio of 6 presented in [2] was tweaked in [6] to a ratio of 5.828, which was the best known ratio until recently. Both algorithms use only a per-edge processing time of $\mathcal{O}(1)$.

Our Contribution. In this paper we present a semi-streaming algorithm that runs in one pass over the input, has a constant per-edge processing time, and that approximates the MWM problem on general graphs with a ratio of 5.585. Therefore it surpasses the known semi-streaming algorithms computing a weighted matching in a single pass. In Section 2

Shadow Matching(G, k)

```
1   M := ∅
2   while input stream is not empty
3      get next input edge y1y2

4      Let g1y1, g2y2 be the edges of M sharing a vertex with y1y2
5      a1g1 := shadow-edge(g1y1, g1)
6      a2g2 := shadow-edge(g2y2, g2)
7      Let a1c1 be the edge of M covering vertex a1
8      Let a2c2 be the edge of M covering vertex a2
9      S := {y1y2, g1y1, a1g1, a1c1, g2y2, a2g2, a2c2}

10     Find an augmenting set A ⊆ S that maximizes r(A) := w(A) − k · w(M(A))
11     if r(A) > 0 then
12        store each edge in M(A) as a shadow-edge of its adjacent edges in A
13        M := (M \ M(A)) ∪ A
```

Figure 1: The algorithm Shadow Matching

we present our algorithm and its main ideas. While the proof of the approximation ratio is found in Section 3, we conclude in Section 4.

2. The Algorithm

In a graph $G = (V, E)$ let two edges be *adjacent* if they have a vertex in common. While M^* denotes a matching of maximum weight in G let in the following M be the matching of G that is currently under consideration by our algorithm. For a set of vertices W we call $M(W)$ to be the set of edges in M covering a vertex in W. Correspondingly, for a set F of edges we denote by $M(F)$ all edges in M that are adjacent to an edge in F. A set of edges in $E \setminus M$ that are pairwise not adjacent we call an *augmenting set*. Throughout the whole paper k denotes a constant greater than 1.

Our algorithm is given in Figure 1. Note at first that each edge in the algorithm is denoted by its endpoints, which is done for the sake of simpler considerations in the

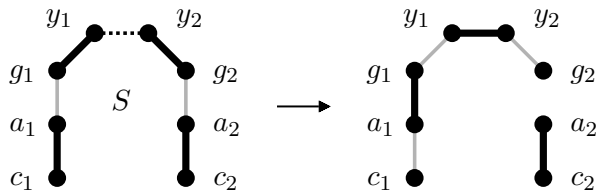


Figure 2: Example of an algorithm's step. Edges in M are shown in bold, shadow-edges appear in grey. y_1y_2 is the actual input edge shown dashed. The algorithm inserts the augmenting set $A = \{y_1y_2, a_1g_1\}$ into M. Therefore the edges $M(A) = \{a_1c_1, g_1y_1, g_2y_2\}$ are removed from M, they become shadow-edges.

following on edges having common vertices. Every edge is well-defined by its endpoints since we assume the input graph G to contain neither multi-edges nor loops.

The general idea of the algorithm is to keep a matching M of G at all times and to decide for each incoming edge y_1y_2 in the input stream if it is inserted into M. This is the case if the weight of y_1y_2 is big compared to the edges already in M sharing a vertex with y_1y_2 and that therefore must be removed from M to incorporate y_1y_2.

This idea so far has already been utilized by one-pass semi-streaming algorithms of Feigenbaum et al.[2] and McGregor[6] seeking a matching in weighted graphs. However, our algorithm differs from the ones in [2] and [6] in fundamental points.

First, if the algorithms in [2] and [6] remove an edge from the actual matching M this is irrevocable. Our new algorithm, by contrast, stores some edges that have been in M in the past but were removed from it. To potentially reinsert them into M the algorithm memorizes such edges under the name of shadow-edges. For an edge xy in M *shadow-edge*(xy, a), $a \in \{x, y\}$, denotes an edge that is stored by the algorithm and shares the vertex a with xy. Every edge xy in M has at most two shadow-edges assigned to it, at most one shadow-edge is assigned to the endpoint x and at most one is assigned to y.

A second main difference is the way of deciding if an edge e is inserted into M or not. In the algorithms of [2] and [6] this decision is based only on the edges in M adjacent to e. Our algorithm takes edges in M as well as shadow-edges in the vicinity of e into account to decide the insertion of e.

Finally, the algorithms of [2] and [6] are limited to the inclusion of the actual input edge into M. By reintegrating shadow-edges our algorithm can insert up to three edges into M within a single step.

Let us take a closer look at the algorithm. As an example of a step of the algorithm, Figure 2 is given. But note that this picture shows only one possible configuration of the set S. Since non-matching edges in S may be adjacent, S may look different.

After reading the actual input edge y_1y_2 the algorithm tags all memorized edges in the vicinity of y_1y_2. This is done in lines 4-8. If an edge is not present the corresponding tag denotes the null-edge, that is, the empty set of weight zero. Thus if for example the endpoint y_2 of the input edge y_1y_2 is not covered by an edge in M, the identifier g_2y_2 denotes a null-edge, as well as its shadow-edge a_2g_2 and the edge a_2c_2. All edges tagged so far are taken into consideration in the remaining part of the loop, they are subsumed to the set S in line 9.

In line 10 all augmenting sets of S are examined. Among these sets the algorithm selects A that maximizes $r(A)$. If $r(A) > 0$ the edges of A are taken into M and the edges in M sharing a vertex with edges in A are removed from M. We say A is *inserted* into M, this is done in line 13.

If an augmenting set A is inserted into M this is always accompanied by storing the removed edges $M(A)$ as shadow-edges of edges in A in line 12. More precisely, every edge e in $M(A)$ is assigned as a shadow-edge to every edge in A that shares a vertex with e. If, as in the example given in Figure 2, $A = \{y_1y_2, a_1g_1\}$, the edge g_1y_1 that is adjacent to both edges in A is memorized under the name shadow-edge(y_1y_2, y_1) as well as under the name shadow-edge(a_1g_1, g_1). a_1c_1 is stored as shadow-edge(a_1g_1, a_1), g_2y_2 as shadow-edge(y_1y_2, y_2). After inserting A, a_2g_2 is not memorized as a shadow-edge assigned to g_2y_2 since g_2y_2 is not an edge in M after the step. That is indicated in Figure 2 by the

disappearance of a_2g_2. However, if a_2g_2 was memorized as a shadow-edge of a_2c_2 before, this will also be the case after inserting A.

It is important to note that there is never an edge in M which is a shadow-edge at the same time: Edges only become shadow-edges if they are removed from M. An edge which is inserted into M is no shadow-edge anymore, since there is no edge in M it could be assigned to as a shadow-edge.

It is easy to see that our algorithm computes a valid matching of the input graph G.

Corollary 2.1. *Throughout the algorithm Shadow Matching(G, k), M is a matching of G.*

Proof. This is true at the start of the algorithm since $M = \emptyset$. Whenever the algorithm modifies M in line 13 it inserts edges that are pairwise not adjacent and removes all edges that are adjacent to the newly inserted ones. Thus M never includes two adjacent edges. ■

Our algorithm may remind of algorithms in [1] and [10] approximating a maximum weighted matching in the RAM model. Starting from some actual matching M in a graph G these algorithms look for short augmentations, that is, connected subgraphs of G having constant size in which edges in M and $E \setminus M$ can be exchanged to increase the weight of the actual matching.

From this point of view our algorithm may suggest itself as it is reasonable to expect the notion of short augmentations to be profitable in the semi-streaming model as well. However, we are unable to use even the basic ideas of proving the approximation ratio in [1] and [10]. As well as the algorithms the proof concept relies on random access to the whole graph, a potential we cannot count on in the semi-streaming model.

Certainly, our algorithm can be considered as a natural extension of the semi-streaming algorithms in [2] and [6] seeking a weighted matching. But the abilities of our algorithm go beyond the insertion of a single edge to the actual matching, the step to which the algorithms in [2] and [6] are limited to. Therefore we have to substantially enhance the proof techniques used therein to attest an improved approximation ratio of our algorithm. This is done in the next section.

3. Approximation Ratio

Consider an augmenting set A which covers the vertices $B \subseteq V$ and let $k > 1$ be some constant. We call $f_{A,k} : V \to \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ an *allocation function* for A if $f_{A,k}(v) = 0$ for all $v \in V \setminus B$ and additionally the following holds:

- $\forall\ ab \in A : f_{A,k}(a) \cdot w(M(a)) + f_{A,k}(b) \cdot w(M(b)) \leq \frac{w(ab)}{k}$
- $\forall\ cd \in M(A) : f_{A,k}(c) + f_{A,k}(d) \geq 1$

If there exists such an allocation function $f_{A,k}$ for an augmenting set A we call A to be *locally k-exceeding*. The intuition here is as follows: If for an augmenting set A we have $w(A) > k \cdot w(M(A))$ we can distribute the weight of the edges in $M(A)$ to the edges of A in such a way that every edge ab in A gets weight of at most $\frac{w(ab)}{k}$ distributed to it. If A satisfies the stronger condition of being locally k-exceeding such a weight distribution can also be done with the additional property that the weight of an edge cd in $M(A)$ is distributed only to edges in A that are adjacent to cd.

Lemma 3.1. *Every augmenting set A that is inserted into M by the algorithm Shadow Matching(G, k) is locally k-exceeding.*

Proof. Since $A \subseteq \{y_1y_2, a_1g_1, a_2g_2\}$ and $r(A) > 0$, $1 \leq |A| \leq 3$. If A consists of only one edge, say y_1y_2, we have for the sum of the weights of the adjacent edges $w(g_1y_1)+w(g_2y_2) \leq \frac{w(y_1y_2)}{k}$ because of the satisfied condition in line 11. In that case the allocation function is $f_{A,k}(y_1) = f_{A,k}(y_2) = 1$ and A is locally k-exceeding.

Let A consist of two edges, say y_1y_2 and a_1g_1. Since every subset of A is an augmenting set as well which is not taken by the algorithm, $r(\{y_1y_2, a_1g_1\}) \geq r(\{y_1y_2\})$ and therefore

$$w(y_1y_2) + w(a_1g_1) - k(w(a_1c_1) + w(g_1y_1) + w(g_2y_2)) \geq w(y_1y_2) - k(w(g_1y_1) + w(g_2y_2))$$

Thus $w(a_1g_1) \geq k \cdot w(a_1c_1)$ and because $r(\{y_1y_2, a_1g_1\}) \geq r(\{a_1g_1\})$ we can deduce similarly $w(y_1y_2) \geq k \cdot w(g_2y_2)$. Hence for the allocation function we can set $f_{A,k}(a_1) = f_{A,k}(y_2) = 1$. Since $r(A) > 0$ we can find appropriate values for $f_{A,k}(g_1)$ and $f_{A,k}(y_1)$, too.

For other configurations of A it can be exploited correspondingly that $r(A) \geq r(A')$ for all subsets A' of A to show the existence of a allocation function for A in a similar way. ∎

Because of Corollary 2.1 we can take the final M of the algorithm as a valid solution for the weighted matching problem on the input graph G. It is immediate that the constant k is crucial for the weight of the solution we get and therefore determines the ratio up to which the algorithm approximates an optimal matching. The main part of the paper is to prove the following theorem which we just state here and which we prove later.

Theorem 3.2. *Let M be a matching constructed by Shadow Matching(G, k), $k > 1$. Then*

$$\frac{w(M^*)}{w(M)} \quad \leq \quad k + \frac{k}{k-1} + \frac{k^3 - k + 1}{k^2}$$

We call G_i the subgraph of G consisting of the first i input edges, M_i denotes the M of the algorithm after completing the while-loop for the ith input edge. An edge xy *prevents* an edge ab if ab is the ith input edge and $xy \in M_i$ shares an endpoint with ab, thus ab is not taken into M by the algorithm. Note that an edge might be prevented by one or two edges. An edge xy *replaces* an edge cd if xy is the ith input edge, xy and cd share a vertex, $cd \in M_{i-1}$, $xy \in M_i$, and therefore $cd \notin M_i$. An edge can replace up to two edges and can be replaced by up to two edges.

Consider an optimal solution $M^* = \{o_1, o_2, \ldots\}$ for the MWM problem of G, $M_i^* := M^* \cap G_i$. The edges $o_1, o_2, \ldots$ in M^* we call *optimal edges*. If $w(M_i) < w(M_i^*)$, some edges of M_i^* must be missing in M_i. There are two possible reasons for the absence of an edge $o_l \in M_i^*$ in M_i. First, there are edges in M_j, $j \leq i$, which prevented o_l. Second, $o_l \in M_j$, $j < i$, is replaced by one or two edges and not reinserted into M afterwards.

In any case we can make edges in $\bigcup_{h \leq i} M_h$ responsible for missing edges of M_i^* in M_i. We charge the weight of an optimal edge o_l to the edges in $\bigcup_{h \leq i} M_h$ that are responsible for the prevention or the removal of o_l. If such a charged edge in M is replaced by other edges its charge is transferred to the replacing edges such that no charge is lost. After all we can sum up the charges of all edges in the final M_m to get $w(M^* \setminus M_m)$.

To bound $w(M_i^* \setminus M_i)$ as a multiple c of $w(M_i)$ if suffices to show that each edge $xy \in M_i$ carries a charge of at most $c \cdot w(xy)$. This technique has been carried out by Feigenbaum et al.[2] and McGregor[6] to estimate the approximation ratios of their semi-streaming algorithms calculating a weighted matching.

We follow the same general idea but need a more sophisticated approach of managing the charge. This is due to two reasons. First, the algorithms of [2] and [6] are limited to a simple replacement step which substitutes one or two edges by a single edge e. That makes

the charge transfer easy to follow since the charges of the substituted edges are transferred completely to e. Our algorithm, by contrast, is able to substitute several edges by groups of edges. The charge to be transferred must be distributed carefully to the replacing edges.

Second, in the algorithms of [2] and [6] the decision whether to insert an input edge into M is determined only by the edges in M adjacent to the input edge. If an optimal edge o is not taken into M the charge can simply be assigned to the at most two edges already in M that are adjacent to o. In our algorithm not only the edges in M that are adjacent to o specify if o is taken into M. In fact, several shadow-edges and other edges in M in the environment of o may codetermine if o is inserted into M. These ambient edges must be taken into account if charge has to be distributed for preventing o.

For our more sophisticated technique of managing the charges we think of every edge $xy \in M$ as being equipped with two values, namely *charge of optimal edge* $coe(xy, x)$ and $coe(xy, y)$, one for every endpoint of xy. $coe(xy, x)$ is the charge that the edge in M^* which is covering the vertex x is charging to xy.

If an edge is removed from M its charges are transfered to the one or two replacing edges. Therefore in addition to its $coe(xy, x)$ and $coe(xy, y)$ every edge $xy \in M$ is equipped with a third value *aggregated charge* $ac(xy)$ which contains charges that xy takes over from edges replaced by xy. We define $T(xy) := coe(xy, x) + coe(xy, y) + ac(xy)$ as the sum of the charges of the edge xy.

During the proof of the following lemma we will explicitly show how the weights of edges in $M_i^* \setminus M_i$ can be charged to the edges in M_i and how these charges are transferred to replacing edges such that particular properties hold.

Lemma 3.3. *Let M_i be the solution found by the algorithm Shadow Matching(G, k), $k > 1$, after reading G_i for $1 \leq i \leq m$. To every edge xy in M_i we can assign three values $coe(xy, x)$, $coe(xy, y)$ and $ac(xy)$, with $T(xy)$ being their sum, such that:*

a) $\sum_{xy \in M_i} T(xy) \geq w(M_i^* \setminus M_i)$

b) $\forall\ xy \in M_i$: $coe(xy, x) \leq k \cdot w(xy)$ *and* $coe(xy, y) \leq k \cdot w(xy)$

c) $\forall\ xy \in M_i$: $ac(xy) \leq \frac{k}{k-1} \cdot w(xy)$

d) $\forall\ xy \in M_i$: $T(xy) \leq \left(k + \frac{k}{k-1} + \frac{k^3-k+1}{k^2}\right) \cdot w(xy)$

Proof. Let y_1y_2 be the ith input edge. If y_1y_2 is an optimal edge that is not taken into M_i by the algorithm we want to charge the weight of y_1y_2 to the edges in M_i that prevented y_1y_2. We first take a look at the different cases that can occur if y_1y_2 is not taken into M_i. We postpone the case in which the set S contains a C_5, i.e., a cycle on five vertices, to the end of this proof. Thus, until further notice S contains no C_5.

If $A = \{y_1y_2, a_1g_1, a_2g_2\}$ is an augmenting set and none of the edges is taken into M_i the condition in line 11 of the algorithm is violated for A and all its subsets. In this case we can split $w(y_1y_2)$ into two partial weights p_1, p_2 and charge p_1 to g_1y_1 and p_2 to g_2y_2 such that the following holds for $x \in \{1, 2\}$:

$$p_x \leq k \cdot w(g_xy_x) \ \text{ and } \ p_x \leq k \cdot (w(g_xy_x) + w(a_xc_x)) - w(a_xg_x) \tag{3.1}$$

Now let one of the edges a_xg_x in A be taken into M_i, w.l.o.g. let this edge be a_1g_1. If y_1y_2 is not inserted into M_i the whole weight of y_1y_2 can be charged to g_2y_2, thus $p_2 = w(y_1y_2)$ and p_2 satisfies condition (3.1) for $x = 2$.

Let a_1g_1 be adjacent to y_1y_2, hence a_1g_1, y_1y_2, and g_1y_1 build a triangle and let $a_2 \neq y_1$ and $a_2 \neq g_1$. If neither y_1y_2 nor a_1g_1 is inserted into M_i we charge $w(y_1y_2)$ as follows: A

part of weight at most $k \cdot w(g_1y_1)$ is charged as p_1 to g_1y_1 such that:

$$p_1 \leq k \cdot w(g_1y_1) \text{ and } y_1y_2,\ g_1y_1, \text{ and } a_1g_1 \text{ build a triangle} \tag{3.2}$$

If a_2g_2 is not inserted into M_i the remaining part of $w(y_1y_2)$ after subtracting p_1 can be charged as p_2 to g_2y_2 satisfying condition (3.1) with $x = 2$. If on the contrary a_2g_2 is taken into M_i there is no remaining part of $w(y_1y_2)$ since then $w(y_1y_2) \leq k \cdot w(g_1y_1)$. In the case that a_1g_1 is inserted into M_i a similar reasoning to the previous one can be applied since now a_1g_1 instead of g_1y_1 is preventing y_1y_2. Therefore the weight charged to a_1g_1 now satisfies a condition similar to (3.2) because $a_1g_1 \in M_i$, g_1y_1, which is now the shadow-edge of a_1g_1, and the prevented edge y_1y_2 form a triangle.

For all other shapes of S (except for the postponed C_5 case) and for all possible augmenting sets it can be shown similarly that $w(y_1y_2)$ of the prevented edge y_1y_2 can be split into two partial weights in such a way that the following generalization holds:

Let $ab \in M_i$ share the vertex a with the ith input edge $o \in M^*$. Let bc be the shadow-edge(ab, b), that is, the shadow-edge assigned to the vertex of ab that is not shared by o. Let cd be the edge in M_i that covers c. $w(o)$ can be split into two partial weights such that for the partial weight p that ab has to take as a charge for preventing o at least one of the following conditions is satisfied:

(I) $p \leq k \cdot w(ab) \leq k \cdot (w(ab) + w(cd)) - w(bc)$
(II) $p \leq k \cdot (w(ab) + w(cd)) - w(bc) \leq k \cdot w(ab)$
(III) $p \leq k \cdot w(ab)$ and ab, input edge o and shadow-edge bc form a triangle.

We start to prove the lemma by induction over the edges inserted into M. More precisely we suppose that the edge y_1y_2 as the ith input edge is inserted into M_i and that before this insertion, i.e., for M_{i-1}, all properties of the lemma are satisfied.

We have to consider two things: First, we have to point out how the charges of the edges in M_{i-1} that y_1y_2 replaces are carried over to y_1y_2 to preserve the properties of the lemma. Second we have to regard the at most two optimal edges that possibly come after y_1y_2 and share a vertex with y_1y_2. If y_1y_2 prevents one or both of these edges we have to show how y_1y_2 is charged by them without violating the lemma.

For the initial step of our induction note that the properties of the lemma hold for the first input edge. For the inductive step let y_1y_2 as the ith input edge be taken into M_i. Thus y_1y_2 is contained in the augmenting set A that is inserted into M. Because of Lemma 3.1 A is locally k-exceeding, hence there exists an allocation function $f_{A,k}$.

Let in the following $x \in \{1, 2\}$. y_1y_2 takes over charges from g_xy_x, the edges it replaces. According to the allocation function $f_{A,k}$ y_1y_2 takes over a $f_{A,k}(y_x)$-fraction of the charges of g_xy_x. In fact, y_1y_2 builds its ac as follows: $ac(y_1y_2) = (coe(g_1y_1, g_1) + ac(g_1y_1)) \cdot f_{A,k}(y_1) + (coe(g_2y_2, g_2) + ac(g_2y_2)) \cdot f_{A,k}(y_2)$. By the induction hypothesis $coe(g_xy_x, g_x) \leq k \cdot w(g_xy_x)$ and $ac(g_xy_x) \leq \frac{k}{k-1} \cdot w(g_xy_x)$. Due to the definition of an allocation function $f_{A,k}(y_1) \cdot w(g_1y_1) + f_{A,k}(y_2) \cdot w(g_2y_2) \leq \frac{w(y_1y_2)}{k}$. Thus $ac(y_1y_2) \leq \frac{k}{k-1} \cdot w(y_1y_2)$ satisfying property c).

Furthermore y_1y_2 takes over charge from $coe(g_xy_x, y_x)$ to its own $coe(y_1y_2, y_x)$, again a $f_{A,k}(y_x)$-fraction of it. If g_xy_x is in M^*, $coe(g_xy_x, y_x) = 0$ and y_1y_2 instead takes over a $f_{A,k}(y_x)$-fraction of $w(g_xy_x)$ as its $coe(y_1y_2, y_x)$ for replacing the optimal edge g_xy_x.

Note that whenever $f_{A,k}(y_x) < 1$, y_1y_2 does not take over all the charge of g_xy_x. However, the definition of the allocation function makes sure that $f_{A,k}(g_x) \geq 1 - f_{A,k}(y_x)$ and that another edge in A covering g_x takes over the remaining charge of g_xy_x. That way no charge can get lost and property a) holds.

Let us check the validity of property b). Right after y_1y_2 was inserted into M and took over the charges as described from g_xy_x it holds that $coe(y_1y_2, y_x) \leq w(y_1y_2)$. That does not suffice to show validity of property b). In fact, there might be an optimal edge o_xy_x coming after y_1y_2 in the input stream covering y_x. In that case $coe(y_1y_2, y_x) = 0$ up to this moment, since there cannot be another optimal edge besides o_xy_x covering y_x. If o_xy_x is not inserted into M, that is, y_1y_2 prevents o_xy_x, y_1y_2 must be charged. By the considerations above we know about the charges that an edge in M has to take because of optimal edges prevented by it. In all three possibilities (I)-(III) the charge y_1y_2 has to include into $coe(y_1y_2, y_x)$ for preventing o_xy_x is at most $k \cdot w(y_1y_2)$, satisfying property b).

It remains to show that property d) holds which bounds the sum of all charges of y_1y_2. The situation is as follows: y_1y_2 is in M and we call the shadow-edge(y_1y_2, y_1) g_1y_1, the shadow-edge(y_1y_2, y_2) g_2y_2. Remember that y_1y_2 took over only a $f_{A,k}(y_x)$-fraction of the charges from g_xy_x. Directly after y_1y_2 was inserted into M and took over the charges from the replaced edges as described property d) holds. We have to consider optimal edges o_xy_x that appear after y_1y_2 in the input stream, are prevented by y_1y_2 and therefore cause charge p_x at $coe(y_1y_2, y_x)$.

As described $ac(y_1y_2)$ is composed of four values, namely fractions of $ac(g_xy_x)$ and $coe(g_xy_y, g_x)$. The value of the fraction of $ac(g_xy_x)$ that is taken over into $ac(y_1y_2)$ we call $ac(g_xy_x) \curvearrowright ac(y_1y_2)$, correspondingly we have $coe(g_xy_x, g_x) \curvearrowright ac(y_1y_2)$. Using that we can separate $T(y_1y_2)$ into two halves as follows

$$\begin{aligned} T(y_1y_2) \;=\; & \Big(coe(y_1y_2, y_2) + ac(g_1y_1) \curvearrowright ac(y_1y_2) + coe(g_1y_1, g_1) \curvearrowright ac(y_1y_2)\Big) + \\ & \Big(coe(y_1y_2, y_1) + ac(g_2y_2) \curvearrowright ac(y_1y_2) + coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2)\Big) \end{aligned}$$

Let us call the upper half $H1$ and the lower one $H2$. We will estimate $H2$ in the following according to the three possible cases for p_1 and show that

$$H2 \;\leq\; \left(k + \frac{1}{k-1} + \frac{1}{k}\right) w(g_2y_2) \cdot f_{A,k}(y_2) \;+\; k \cdot w(y_1y_2) \qquad (*)$$

We will see later that it suffices to show that if neither $H2$ violates inequality (*) nor $H1$ violates a corresponding inequality, property d) holds for y_1y_2.

Charge p_1 coming from o_1y_1 satisfies (I)
Let g_2z_2 be an edge in M covering g_2. We can bound p_1 because of property (I)

$$p_1 \leq k \cdot w(y_1y_2) \leq k \cdot (w(y_1y_2) + w(g_2z_2)) - w(g_2y_2) \qquad (3.3)$$

We call the shadow-edge g_2y_2 of y_1y_2 *overloaded* if we have $coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2) > w(g_2y_2) \cdot f_{A,k}(y_2)$. For a shadow-edge uv we say that uv *fingers* v if uv covers v and v is not the vertex that uv shares with the edge in M it is assigned to. For example the shadow-edge g_2y_2, which is assigned to y_1y_2, fingers g_2 but not y_2. A shadow-edge uv is *prepared* if for the edge uw in M that uv is assigned to $coe(uw, w) = 0$. So in the present example g_2y_2 is prepared if $coe(y_1y_2, y_1) = 0$.

If $p_1 \leq k \cdot w(y_1y_2) - f_{A,k}(y_2) \cdot w(g_2y_2)$ or if g_2y_2 is not overloaded, we can simply add p_1 to $coe(y_1y_2, y_1)$ and $H2$ satisfies (*). Otherwise we do a *charge transfer* as follows: We reduce $coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2)$ to $r := \max\{coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2) - (k-1) \cdot w(g_2z_2), 0\}$ and add a value of $coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2) - r$ to $coe(g_2z_2, g_2)$, thus no charge is lost.

It is important to see that this increasing of $coe(g_2z_2, g_2)$ does not violate the properties of the lemma for g_2z_2: We know that $coe(g_2z_2, z_2) \leq k \cdot w(g_2z_2)$ and $ac(g_2z_2) \leq \frac{k}{k-1} \cdot w(g_2z_2)$.

If before the charge transfer $coe(g_2z_2, g_2) = 0$, after the transfer $T(g_2z_2)$ cannot exceed $(k + \frac{k}{k-1} + \frac{k^3-k+1}{k^2}) \cdot w(g_2z_2)$.

For the other case, i.e., that $coe(g_2z_2, g_2) > 0$ before the charge transfer we need a few considerations. In fact, we will show that for every vertex v at every moment of the algorithm at most one shadow-edge fingers v, is overloaded, and prepared at the same time:

Assume that uv is the first shadow-edge created by the algorithm that is fingering v and that is overloaded and prepared. This can only be the case if uv in M gets replaced by uw and possibly vs. uv as a shadow-edge of uw is now fingering v and it is overloaded and prepared. Right after the replacement $coe(vs, v) \leq w(vs)$. As long as no charge of $coe(uv, v) \curvearrowright ac(uw)$ is transferred to an edge in M covering v, for every edge vq in M covering v $coe(vq, v) \leq w(vq)$. Such an edge vq cannot be turned into a shadow-edge fingering v and being overloaded. A second overloaded shadow-edge fingering v can only be created by replacing an edge vr with $coe(vr, v) > w(vr)$, that can only occur if uw transfers charge to vr. However, uw only transfers charge to vr if it prevents an optimal edge. After that $coe(uw, w) > 0$ and uv is not prepared anymore. This shows that a prepared and overloaded shadow-edge fingering v can only be created if the at most one previously prepared and overloaded shadow-edge fingering v lost its status as being prepared.

Now we can come back to the case $coe(g_2z_2, g_2) > 0$. We can assume that g_2z_2 as part of the augmenting set A' replaced the edges d_2g_2 and t_2z_2. g_2z_2 took over a $f_{A',k}(g_2)$-fraction of the charges from d_2g_2. Since $coe(d_2g_2, g_2) \leq k \cdot w(d_2g_2)$ before the replacement of d_2g_2, we have $coe(g_2z_2, g_2) \leq f_{A',k}(g_2) \cdot k \cdot w(d_2g_2)$ after the replacement. By the definition of an allocation function it follows $coe(g_2z_2, g_2) \leq w(g_2z_2) - f_{A',k}(z_2) \cdot k \cdot w(t_2z_2)$. After our charge transfer of weight at most $(k-1) \cdot w(g_2z_2)$ from $coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2)$ to $coe(g_2z_2, g_2)$, it holds that $coe(g_2z_2, g_2) \leq k \cdot (w(g_2z_2) - f_{A',k}(z_2) \cdot w(t_2z_2))$. Therefore the charges of g_2z_2 satisfy an inequality corresponding to (*), thus property d) cannot be violated for g_2z_2.

Now the above considerations are important: We know that no shadow-edge besides g_2y_2 that is fingering g_2 is prepared and overloaded. Thus no further charge transfer to $coe(g_2z_2, g_2)$ can occur violating the properties of the lemma for g_2z_2.

After transferring a part of $coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2)$ as described we have $coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2) \leq \max\{k \cdot f_{A,k}(y_2) \cdot w(g_2y_2) - (k-1) \cdot w(g_2z_2), 0\}$. We add p_1 to $coe(y_1y_2, y_1)$ and can evaluate $H2$: We have $coe(y_1y_2, y_1) = p_1 \leq k \cdot w(y_1y_2)$ because of (3.3) and $ac(g_2y_2) \curvearrowright ac(y_1y_2) \leq f_{A,k}(y_2) \cdot w(g_2y_2) \cdot \frac{k}{k-1}$ by the induction hypothesis. Since $w(g_2z_2) \geq \frac{w(g_2y_2)}{k}$ because of (3.3) we can estimate $H2$ as being bounded as in inequality (*).

Charge p_1 coming from o_1y_1 satisfies (II)
This case is very similar to the previous one with the only difference that $w(g_2z_2) \leq \frac{w(g_2y_2)}{k}$ and we use $p_1 \leq k \cdot (w(y_1y_2) + w(g_2z_2)) - w(g_2y_2)$. All other considerations remain the same and that results in the very same estimation for $H2$.

Charge p_1 coming from o_1y_1 satisfies (III)
In this case $o_1 = g_2$ since the input edge o_1y_1, the edge $y_1y_2 \in M$ and the shadow-edge g_2y_2 form a triangle. Since g_2y_1 is an optimal edge, before its arrival $coe(g_2y_2, g_2) \curvearrowright ac(y_1y_2) = 0$. So y_1y_2 can take a charge of $p_1 \leq k \cdot w(y_1y_2)$ as its $coe(y_1y_2, y_1)$ and $H2$ satisfies (*).

We can handle the charge p_1 in every possible case such that $H2$ satisfies (*). With a symmetric argumentation we can show that $H1$ satisfies a corresponding inequality. Using

that $f_{A,k}(y_1) \cdot w(g_1y_1) + f_{A,k}(y_2) \cdot w(g_2y_2) \leq \frac{w(y_1y_2)}{k}$ we get validity of property d) since

$$T(y_1y_2) = H1 + H2 \leq \left(k + \frac{k}{k-1} + \frac{k^3 - k + 1}{k^2}\right) \cdot w(y_1y_2)$$

It remains to consider the postponed situation in which S contains a C_5. This can only be the case if $a_1 = a_2$. If $y_1y_2 \in M^*$ as the ith input edge is prevented but one of the edges a_1g_1, a_2g_2 is inserted into M_i the edge g_1y_1 (g_2y_2, respectively) can be charged with $w(y_1y_2)$ and this charge satisfies condition (I) or (II).

The last possibility is the one in which $a_1 = a_2$ and no augmenting set is inserted into M at all. Assume now that this is the case, thus the situation is as follows: g_1y_1 and g_2y_2 are in M, $a_1g_1 = \text{shadow-edge}(g_1y_1, g_1)$ and $a_2g_2 = \text{shadow-edge}(g_2y_2, g_2)$. g_1y_1 took over a $f_{A',k}(g_1)$-fraction of the charges from a_1g_1 when replacing it, g_2y_2 took over a $f_{A'',k}(g_2)$-fraction of the charges from a_2g_2. Since $a_1 = a_2$ it is also $c_1 = c_2$.

Let w.l.o.g. $f_{A',k}(g_1) \cdot w(a_1g_1) \geq f_{A'',k}(g_2) \cdot w(a_2g_2)$. It suffices to consider y_1y_2 as an optimal edge since otherwise no charge must be assigned if y_1y_2 is prevented and the properties of the lemma hold further on.

Prior the arrival of y_1y_2, $coe(g_1y_1, y_1) = coe(g_2y_2, y_2) = 0$, thus a_1g_1 and a_2g_2 are both prepared and fingering a_1. If $coe(a_2g_2, a_2) \curvearrowright ac(g_2y_2) = f_{A'',k}(g_2) \cdot w(a_2g_2) + X$ for $X > 0$, a_2g_2 is overloaded, thus $coe(a_1g_1, a_1) \curvearrowright ac(g_1y_1) \leq f_{A',k}(g_1) \cdot w(a_1g_1)$ since a_1g_1 cannot be overloaded as well. X cannot be greater than $(k-1) \cdot f_{A'',k}(g_2) \cdot w(a_2g_2)$, therefore we can transfer a charge of weight X from $coe(a_2g_2, a_2) \curvearrowright ac(g_2y_2)$ to $coe(a_1g_1, a_1) \curvearrowright ac(g_1y_1)$, a_1g_1 might get overloaded, a_2g_2 is not overloaded anymore.

After this transfer of charge, or if no transfer was necessary because $X \leq 0$, we have $coe(a_2g_2, a_2) \curvearrowright ac(g_2y_2) \leq f_{A'',k}(g_2) \cdot w(a_2g_2)$. Thus $coe(g_2y_2, y_2)$ can take a charge of $k \cdot w(g_2y_2)$ without violating the properties of the lemma since in that case $coe(g_2y_2, y_2)$, $coe(a_2g_2, a_2) \curvearrowright ac(g_2y_2)$ and $ac(a_2g_2) \curvearrowright ac(g_2y_2)$ still satisfy an inequality corresponding to (*). If no augmenting set is inserted into M, $w(y_1y_2) \leq \min\{k \cdot (w(g_1y_1) + w(g_2y_2)), k \cdot (w(g_1y_1) + w(a_1c_1)) - w(a_1g_1) + k \cdot w(g_2y_2)\}$. Therefore the partial weight of y_1y_2 that g_1y_1 has to take as charge for preventing y_1y_2 satisfies the properties (I) or (II) and can be handled as described before.

We showed that the properties a)-d) of the lemma hold when y_1y_2 replaces and prevents edges. In the very same way the validity of the properties can be shown for the edges a_1g_1 and/or a_2g_2 that are possibly taken into M at the same time as y_1y_2. ∎

Using Lemma 3.3 we can prove our main theorem.

Proof of Theorem 3.2: Let M be the final M_m. $w(M^*) = w(M^* \cap M) + w(M^* \setminus M)$. Because for an edge $xy \in M^* \cap M$ we have $coe(xy, x) = coe(xy, y) = 0$, we can write

$$w(M^* \setminus M) \leq \sum_{xy \in M^* \cap M} \frac{k}{k-1} \cdot w(xy) + \sum_{uv \in M \setminus M^*} T(uv)$$

That results in $w(M^*) \leq \left(k + \frac{k}{k-1} + \frac{k^3-k+1}{k^2}\right) \cdot w(M)$. ∎

The term describing the approximation ratio of our algorithm reaches its minimum for k being around 1.717, that yields a ratio of 5.585. It is easy to see that the algorithm does not exceed the space restrictions of the semi-streaming model: It needs to memorize the edges of M, for each of those at most two shadow-edges, thus it suffices to store a linear number

of edges. The time required to handle a single input edge is determined by the size of S. Since S is of constant size, a single run of the while loop, including the enumeration and comparison of all possible augmenting sets of S, can be done in constant time. Therefore the algorithm needs a per-edge processing time of $\mathcal{O}(1)$ and is content with a single pass over the input.

4. Conclusion

We presented a semi-streaming algorithm calculating a weighted matching in a graph G. Our algorithm achieves an approximation ratio of 5.585 and therefore surpasses all previous algorithms for the maximum weighted matching problem in the semi-streaming model. In addition to the edges of an actual matching M the algorithm memorizes some more edges of G, the so called shadow-edges. For each input edge e, the subgraph S made up of e and of shadow-edges and edges of M in the vicinity of e is examined. If a certain gain in the weight of M can be made, matching and non-matching edges in S are exchanged.

The subgraph S investigated by our algorithm for each input edge consists of at most seven edges. It is reasonable to assume that by examining bigger subgraphs the approximation ratio can be enhanced further. Therefore we believe that extending our approach will lead to improved semi-streaming algorithms computing a weighted matching.

References

[1] D. E. Drake Vinkemeier, S. Hougardy. A linear-time approximation algorithm for weighted matchings in graphs. ACM Transactions on Algorithms (TALG) 1(1), 107-122, 2005.

[2] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. ICALP 2004, In: LNCS 3142, 531-543, 2004.

[3] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph Distances in the Streaming Model: The Value of Space. In: Proc. ACM-SIAM SODA (2005): 745-754, 2005.

[4] H.N. Gabow. Data structures for weighted matchings and nearest common ancestors with linking. In: Proc. ACM-SIAM SODA (1990), 434-443, 1990.

[5] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In: External Memory Algorithms, DIMACS Ser. in Discr. Math. and Theoret. Comp. Science, 50:107-118, 1999

[6] A. McGregor. Finding Graph Matchings in Data Streams. APPROX and RANDOM 2005, In: LNCS 3624, 170-181, 2005.

[7] R.H. Möhring, M. Müller-Hannemann. Complexity and modeling aspects of mesh refinement into quadrilaterals. Algorithmica 26, 148-171, 2000.

[8] B. Monien, R. Preis, and R. Diekmann. Quality matching and local improvement for multilevel graph-partitioning. Paral. Comput. 26, 1609-1634, 2000.

[9] S. Muthukrishnan. Data streams: Algorithms and applications. 2003. Available at http://athos.rutgers.edu/∼muthu/stream-1-1.ps

[10] S. Pettie, P. Sanders. A simpler linear time 2/3-ε approximation for maximum weight matching. Information Processing Letters 91(6), 271-276, 2004.

[11] R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. STACS 1999, In: LNCS 1563, 259-269, 1999.

[12] A. Schrijver. Combinatorial optimization: polyhedra and efficiency. Springer, Heidelberg, 2003.

[13] M. Zelke. Optimal Per-Edge Processing Times in the Semi-Streaming Model. Information Processing Letters 104(3), 106-112, 2007.

Index of authors

P. Albert 39
A. Ambainis 49
A. Ballier 61
V. Bárány 385
L. Bienvenu 73
A. Björklund 85
M. Bläser 97
H. Bodlaender 657
V. Bonifaci 109
P. Bouyer 121
P. Briest 133
J. Brody 145
V. Chakaravarthy 157
A. Chakrabarti 145
C. Chen 169
É. Colin de Verdière 181
A. F. Cook IV 193
M. Crochemore 11, 205
M. Damian 217
S. Datta 229
M. Dietzfelbinger 241
J.-F. Dufourd 253
B. Durand 61
T. Erlebach 265, 277
J. Esparza 289
D. Fischer 301
R. Flatland 217
D. Freedman 169
T. Ganzow 313
W. Gelade 325
C. Glaßer 337
E. Grädel 301
T. Hagerup 265
E. Hemaspaandra 349
V. T. Hoang 361
M. Hoefer 133
C. Hoffmann 97
M. Hoffmann 277
T. Husfeldt 85
L. Ilie 11
C. Iliopoulos 205
K. Jansen 265
E. Jeandel 61
A. Jeż 373
L. Kaiser 301, 385
I. Kanj 397, 409
J.-Y. Kao 421
P. Kaski 85
S. Kiefer 289
J. Kinne 433
F. Klaedtke 445
M. Koivisto 85
A. Kojevnikov 457
P. Korteweg 109
D. Krizanc 277
P. Krysta 133
M. Kubica 205
R. Kulkarni 229
D. Kuske 467
S. Laue 469
P. Lehtinen 645
A. Li 491
Z. Lotker 503
S. Lovett 515
P. Lu 527
M. Luttenberger 289
A. Marchetti-Spaccamela 109
N. Markey 121
E. Mayordomo 39
J. Mestre 539
U. Meyer 551
M. Mihal'ák 277
M. Minzlaff 265
M. Mishna 561

P. Moser 39
A. Muchnik 73
F. Murlak 573
F. Neven 325
S. Nikolenko 457
A. Okhotin 373
J. O'Rourke 217
J. Ouaknine 121
B. Patt-Shamir 503
M. J. Pelsmajer 397
S. Perifel 39
L. Perković 409
J.-É. Pin 585
M. S. Rahman 205
R. Raman 277
S. Ramaswami 217
D. Rawitz 503
B. Rosgen 597
J. E. Rowe 241
S. Roy 157, 229
S. Rubin 313, 385
C. Saha 609
J. Sakarovitch 621
M. Schaefer 397
H. Schmitz 337
P. Schnoebelen 121
H. Schnoor 349
A. Schrijver 181
T. Schwentick 17
V. Selivanov 337
J. Shallit 421
A. Shen 73
P. Silva 585
R. de Souza 621
L. Stougie 109
W.-K. Sung 361
T. Thierauf 633
D. van Melkebeek 433
J. van Rooij 657
A. Valmari 645
N. Vereshchagin 73
F. Wagner 633
T. Waleń 205
I. Wegener 241
C. Wenk 193
P. Woelfel 241
A. Wolff 265
J. Worrell 121
G. Xia 397
M. Xia 491
Z. Xu 421
M. Yannakakis 19
C. Yu 527
M. Zabrocki 561
M. Zelke 669