Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 1-7
www.stacs-conf.org

FOREWORD

SUSANNE ALBERS [1] AND JEAN-YVES MARION [2]

[1] Department of Computer Science, University of Freiburg
E-mail address: salbers@informatik.uni-freiburg.de

[2] Loria and ENS des Mines de Nancy
E-mail address: jean-yves.marion@loria.fr

The Symposium on Theoretical Aspects of Computer Science (STACS) is held alternately in France and in Germany. The conference of February 26-28, 2009, held in Freiburg, is the 26th 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), Aachen (2007), and Bordeaux (2008).

The interest in STACS has remained at a high level over the past years. The STACS 2009 call for papers led to over 280 submissions from 41 countries. Each paper was assigned to three program committee members. The program committee held a two-week electronic meeting at the beginning of November and selected 54 papers. As co-chairs of the program committee, we would like to sincerely thank its members and the many external referees for their valuable work. The overall very high quality of the submissions made the selection a difficult task.

We would like to express our thanks to the three invited speakers, Monika Henzinger, Jean-Éric Pin and Nicole Schweikardt, for their contributions to the proceedings.

Special thanks are due to A. Voronkov for his EasyChair software (www.easychair.org). Moreover we would like to thank Sonja Lauer for preparing the conference proceedings and continuous help throughout the conference organization.

For the second time this year's STACS proceedings are published in electronic form. A printed version was also available at the conference, with ISBN 978-3-939897-09-5. 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).

STACS 2009 received funds from the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) and the University of Freiburg. We thank them for their support!

December 2008

Susanne Albers
Jean-Yves Marion

Conference organization

STACS 2009 was organized by the Chair of Algorithms and Complexity at the University of Freiburg under the auspices of the Special Interest Group for Theoretical Computer Science of the Gesellschaft für Informatik (GI).

Members of the Program Committee

Susanne Albers	*University of Freiburg*
Andris Ambainis	*University of Latvia*
Philippe Baptiste	*École Polytechnique & CNRS LIX*
Holger Bast	*MPI Saarbrücken*
Patricia Bouyer	*CNRS & ENS Cachan*
Martin Dietzfelbinger	*TU Ilmenau*
Zoltán Ésik	*University of Szeged & University of Tarragona*
Christiane Frougny	*University of Paris 8 & LIAFA CNRS*
Leonid Libkin	*University of Edinburgh*
Meena Mahajan	*Institute of Mathematical Sciences Chennai*
Jean-Yves Marion	*Nancy-Université & Loria*
K. Narayan Kumar	*Chennai Mathematical Insitute*
Friedrich Otto	*University of Kassel*
Joël Ouaknine	*University of Oxford*
Harald Räcke	*University of Warwick*
Eric Rémila	*CNRS & University of St-Etienne*
Adi Rosén	*CNRS & University of Paris 11*
Martin Skutella	*TU Berlin*
Jeremy Spinrad	*Vanderbilt University*
Kavitha Telikepalli	*IISc Bangalore*
Thomas Wilke	*University of Kiel*
Philipp Woelfel	*University of Calgary*

Members of the Organizing Committee

Susanne Albers
Gero Greiner
Tobias Jacobs
Sonja Lauer
Alexander Souza

External reviewers

Karen Aardal
Dimitris Achlioptas
Anil Ada
Anna Adamaszek
Isolde Adler
Manindra Agrawal
Oswin Aichholzer
Miklós Ajtai
Deepak Ajwani
Ali Akhavi
Eric Allender
Kazuyuki Amano
Giuseppe Amato
Maxime Amblard
Amihood Amir
Luis Antunes
Krzysztof Apt
Marcelo Arenas
Vikraman Arvind
Arash Asadpour
Albert Atserias
Franz Aurenhammer
Chen Avin
Yossi Azar
Nikhil Bansal
Pablo Barceló
Richard Baron
Cédric Bastoul
Sylvain Béal
Luca Becchetti
Florent Becker
József Békési
Michael Benedikt
Sergey Bereg
Petra Berenbrink
Martin Berger
Jean Berstel
Nathalie Bertrand
Dietmar Berwanger
Peter Biro
Somenath Biswas
Henrik Björklund
Bruno Blanchet
Markus Bläser
Johannes Blömer
Achim Blumensath
Hans-Joachim Böckenhauer
Olivier Bodini
Manuel Bodirsky
Hans Bodlaender
Andrej Bogdanov
Bernard Boigelot
Beate Bollig
Guillaume Bonfante
Paul Bonsma
Henning Bordihn
Olivier Bournez
Octave Boussaton
Andreas Brandstädt
Thomas Brihaye
Andrei Bulatov
Jérôme Buzzi
Sergio Cabello
Diletta Romana Cacciagrano
Alberto Caprara
Ioannis Caragiannis
Arnaud Carayol
Olivier Carton
Catarina Carvalho
Gianpiero Cattaneo
Marjan Celikik
Amit Chakrabarti
Deeparnab Chakrabarty
Sourav Chakraborty
Tanmoy Chakraborty
Jérémie Chalopin
Erin Chambers
Krishnendu Chatterjee
Jen-Yeu Chen
Victor Chepoi
Christian Choffrut
Marek Chrobak
Ken Clarkson
Julien Clément
Raphaël Clifford
Bob Coecke
Johanne Cohen
Amin Coja-Oghlan
Sébastien Collette
Vincent Conitzer
Colin Cooper
Derek Corneil
José Correa
Véronique Cortier
Marie-Christine Costa
Bruno Courcelle
Giovanni Di Crescenzo
Maxime Crochemore
Mary Cryan
László Csirmaz
Péter Csorba
Liliana Cucu
Artur Czumaj
Víctor Dalmau
Carsten Damm
Bireswar Das
Jürgen Dassow
Samir Datta
Alain Daurat
Matei David
Anuj Dawar
Brian Dean
Daniel Delling
Marianne Delorme
François Denis
Amit Deshpande
Alin Deutsch
Florian Diedrich
Volker Diekert
Michael Dom
Frederic Dorn
Rodney G. Downey
Daniel Dressler
Manfred Droste
Deepak D'Souza
Donglei Du
Marie Duflot
Jacques Duparc
Bruno Durand
Jérôme Durand-Lose
Christoph Dürr
Zeev Dvir
Bettina Eick
Khaled Elbassioni
Michael Elkin
Faith Ellen
Robert Elsässer
Matthias Englert
Leah Epstein
Thomas Erlebach
Kousha Etessami
Martin Farach-Colton
Hugues Fauconnier
John Fearnley

Tomás Feder
Mike Fellows
Daniel Fernholz
Guillaume Fertin
Abraham Flaxman
Paola Flocchini
Fedor Fomin
Pierre Fraigniaud
Kimmo Fredriksson
Tobias Friedrich
Alan Frieze
Bernhard Fuchs
Stefan Funke
Anahí Gajardo
Jérôme Galtier
Yong Gao
Bernd Gärtner
Floris Geerts
Hugo Gimbert
Mathieu Giraud
Christian Glaßer
Xavier Goaoc
Andreas Goerdt
Leslie Ann Goldberg
Rob Goldblatt
Michael Goodrich
Navin Goyal
Erich Grädel
Jens Gramm
Etienne Grandjean
Fabrizio Grandoni
Catherine Greenhill
Serge Grigorieff
Alexander Grigoriev
Martin Grohe
André Gronemeier
Philippe de Groote
Roberto Grossi
Christoph Haase
Torben Hagerup
MohammadTaghi Hajiaghayi
Yijie Han
Edda Happ
Ramesh Hariharan
Tero Harju
Tobias Harks
Frédéric Havet
Edith Hemaspaandra
Miki Hermann
Danny Hermelin
Ulrich Hertrampf
William Hesse
Volker Heun
Lisa Higham
Benjamin Hiller
Denis Hirschfeldt
Mika Hirvensalo
John Hitchcock
Petr Hliněný
Thanh Minh Hoang
Michael Hoffmann
Markus Holzer
Florian Horn
Peter Hoyer
Mathieu Hoyrup
Xiuzhen Huang
Anna Huber
Mark Huber
Falk Hüffner
Paul Hunter
Thore Husfeldt
David Ilcinas
Lucian Ilie
Keiko Imai
Csanád Imreh
François Irigoin
Szabolcs Iván
Satoru Iwata
Riko Jacob
Rahul Jain
Emmanuel Jeandel
Peter Jeavons
Galina Jiraskova
Pushkar Joglekar
Daniel Johannsen
Vincent Jost
Ruben Juarez
Stasys Jukna
Marcin Jurdziński
Tomasz Jurdziński
Kanela Kaligosi
Mihyun Kang
Juhani Karhumäki
Jarkko Kari
Juha Kärkkäinen
Andreas Karrenbauer
Matya Katz
Wayne Kelly
Michael Kerber
Roni Khardon
James King
Daniel Kirsten
Hartmut Klauck
Rolf Klein
Shmuel Klein
Ines Klimann
Christian Knauer
Johannes Köbler
Ekkehard Köhler
Jochen Könemann
Nitish Korula
Dieter Kratsch
Stefan Kratsch
Robi Krauthgamer
Steve Kremer
Danny Krizanc
Sven Krumke
Piotr Krysta
Gregory Kucherov
Fabian Kuhn
Amit Kumar
Michal Kunz
Orna Kupferman
Klaas Ole Kürtz
Agi Kurucz
Piyush Kurur
Dietrich Kuske
Martin Kutrib
Oded Lachish
Klaus-Jörn Lange
Martin Lange
Sophie Laplante
Benoit Larose
François Laroussinie
Ranko Lazic
Emmanuelle Lebhar
Thierry Lecroq
Troy Lee
Arnaud Lefebvre
Hanno Lefmann
Jérôme Leroux
Lucas Létocart
Jian Li
Zongpeng Li
Nutan Limaye
Gerhard Lischke
Kamal Lodaya
Christof Löding
Markus Lohrey
Satyanarayana Lokam
Daniel Lokshtanov
Vadim Lozin
Marco Lübbecke

Michael Luttenberger
Laurent Lyaudet
Avner Magen
Frédéric Magniez
Mohammad Mahdian
Veli Mäkinen
Andreas Maletti
Rémy Malgouyres
Sebastian Maneth
Yishay Mansour
Alberto Marchetti-Spaccamela
Maurice Margenstern
Nicolas Markey
Petar Markovic
Dániel Marx
Martín Matamala
Domagoj Matijevic
Yuri Matiyasevich
Kaczmarek Matthieu
Dillon Mayhew
Richard Mayr
Frédéric Mazoit
Jacques Mazoyer
Andrew McGregor
Pierre McKenzie
Nicole Megow
Dieter van Melkebeek
Wolfgang Merkle
Stephan Mertens
Hartmut Messerschmidt
Julián Mestre
Antoine Meyer
Dominique Michelucci
Peter Bro Miltersen
Sounaka Mishra
Michael W. Mislove
Michael Mitzenmacher
Michael Molloy
Fabien de Montgolfier
Hannes Moser
Larry Moss
Elchanan Mossel
Jean-Yves Moyen
Indraneel Mukherjee
Partha Mukhopadhyay
Madhavan Mukund
Ian Munro
Filip Murlak
Viswanath Nagarajan
Giacomo Nannicini
Meghana Nasre
Rouven Naujoks
Gonzalo Navarro
Ashwin Nayak
Yakov Nekrich
Frank Neumann
Phong Nguyen
Prajakta Nimbhorkar
Hidenosuke Nishio
Damian Niwiński
Dirk Nowotka
Ryan O'Donnell
Nicolas Ollinger
Rafail Ostrovsky
Michio Oyamaguchi
Leszek Pacholski
Rasmus Pagh
Laxmi Parida
Mihăi Patraşcu
Boaz Patt-Shamir
Romain Péchoux
Andrzej Pelc
Simon Perdrix
Marc E. Pfetsch
Christophe Picouleau
Brigitte Pientka
Jean-Éric Pin
Nir Piterman
Ely Porat
Lars Prädel
Kirk Pruhs
Daniel Prusa
Simon Puglisi
Evangelia Pyrga
Jaikumar Radhakrishnan
Tomasz Radzik
Mathieu Raffinot
Daniel Raible
Liva Ralaivola
C. R. Ramakrishnan
Venkatesh Raman
R. Ramanujam
M. V. Panduranga Rao
B. V. Raghavendra Rao
Damien Regnault
Daniel Reidenbach
Rüdiger Reischuk
Christian Reitwießner
Giuseppina Rindone
Romeo Rizzi
Julien Robert
Andrei Romashchenko
Peter Rossmanith
Günter Rote
Thomas Rothvoss
Jean-Baptiste Rouquier
Sambuddha Roy
Peter Ryan
Wojciech Rytter
Amin Saberi
Rei Safavi-Naini
Mohammad Salavatipour
Arnaud Sangnier
Rahul Santhanam
Luigi Santocanale
Ramprasad Saptharishi
Gabor Sarkozy
Jayalal Sarma
Srinivasa Rao Satti
Ravi Vijaya Satya
Martin Sauerhoff
Saket Saurabh
Gabriel Scalosub
Francesco Scarcello
Nicolas Schabanel
Mathias Schacht
Guido Schäfer
Dominik Scheder
Christian Scheideler
Sven Schewe
Christian Schindelhauer
Henning Schnoor
Uwe Schöning
Anna Schulze
Pascal Schweitzer
Thomas Schwentick
Maria Grazia Scutellà
Michael Segal
Pranab Sen
Géraud Sénizergues
Olivier Serre
Rocco Servedio
Jeffrey Shallit
Ronen Shaltiel
Vitaly Shmatikov
Amin Shokrollahi
Anastasios Sidiropoulos
Somnath Sikdar
Pedro V. Silva
Hans Simon
René Sitters
Isabelle Sivignon
Alex Slivkins

Robert Sloan
Shakhar Smorodinsky
William F. Smyth
Christian Sohler
Philippe Solal
Jonathan Sorenson
Alexander Souza
Srikanth Srinivasan
Heiko Stamer
Ian Stark
Rob van Stee
Damien Stehlé
Frank Stephan
David Steurer
Nicolás Stier-Moses
Howard Straubing
Karol Suchan
S. P. Suresh
Maxim Sviridenko
Andrzej Szepietowski
Thomas Thierauf
Eric Thierry
Dimitrios Thilikos
Wolfgang Thomas
Stéphan Thomassé
Mikkel Thorup
Sébastien Tixeuil
Anthony Widjaja To
Ioan Todinca
Jacobo Torán
Corinne Touati
Luca Trevisan
Mirosław Truszczyński
Denis Trystram
Madhur Tulsiani
Andrew Turpin
Andy Twigg
Chris Umans
Christian Urban
Sándor Vágvölgyi
Kasturi Varadarajan
Vinodchandran Variyam
Éric Colin de Verdière
Nikolay Vereshchagin
Jose Verschae
Laurent Vigneron
Dan Vilenchik
V. Vinay
Emanuele Viola
Sundar Vishwanathan
Paul Vitanyi
Berthold Vöcking
Heribert Vollmer
Jan Vondrak
Vladimir V'yugin
Magnus Wahlström
Yoshiko Wakabayashi
Igor Walukiewicz
Rolf Wanka
Bogdan Warinschi
John Watrous
Pascal Weil
Gera Weiss
Andreas Wiese
Gerhard Woeginger
Ronald de Wolf
Pierre Wolper
David Woodruff
James Worrell
Mihalis Yannakakis
Ke Yi
Sheng Yu
Luca Zamboni
Mariano Zelke
Wiesław Zielonka
Uri Zwick

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 13–30
www.stacs-conf.org

A COMPARISON OF TECHNIQUES FOR SAMPLING WEB PAGES

EDA BAYKAN [1] AND MONIKA HENZINGER [1,2] AND STEFAN F. KELLER [3] AND SEBASTIAN DE CASTELBERG [3] AND MARKUS KINZLER [3]

[1] Ecole Polytechnique Fédérale de Lausanne (EPFL)
IC LTAA Station 14 CH-1015 Lausanne Switzerland
E-mail address: eda.baykan@epfl.ch

[2] Google Switzerland

[3] University of Applied Science Rapperswil, Switzerland

Abstract. As the World Wide Web is growing rapidly, it is getting increasingly challenging to gather representative information about it. Instead of crawling the web exhaustively one has to resort to other techniques like sampling to determine the properties of the web. A uniform random sample of the web would be useful to determine the percentage of web pages in a specific language, on a topic or in a top level domain. Unfortunately, no approach has been shown to sample the web pages in an unbiased way. Three promising web sampling algorithms are based on random walks. They each have been evaluated individually, but making a comparison on different data sets is not possible. We directly compare these algorithms in this paper. We performed three random walks on the web under the same conditions and analyzed their outcomes in detail. We discuss the strengths and the weaknesses of each algorithm and propose improvements based on experimental results.

Introduction

The World Wide Web is a rich source of information about the world but very little information is known about the web itself. We do not know what percentage of web pages are in a specific language or on a topic or in a top level domain. There are estimates on what percentage of web pages change per day [7, 8] but they depend on how deeply the sites were crawled. Trying to determine these statistics based on exhaustive enumeration of the web is not feasible because of its size and its rapidly changing nature. However, a uniform random sample of the web[1] would provide answers to many of the above questions and repeated sampling would also allow to monitor changes in the web's composition.

1998 ACM Subject Classification: G.2.2 Graph Algorithms, H.2.8 Data Mining.
Key words and phrases: Random walks, sampling web pages.
Preliminary results of this paper were presented at IIWeb 2006 Workshop.

[1]We refer to a uniform random sample of the web as the uniform random sample of the web pages *not* of the graph structure of the web.

In the literature there are four major approaches for sampling web pages: Lawrence and Giles [12] tested random IP addresses to determine characteristics of hosts. However, it leaves the question open how to deal with multiple hosts sharing the same IP address or hosts being spread over multiple IP addresses. Additionally, it is not clear how to sample the web pages accessible at a given IP address. Thus, this approach samples IP addresses, but not web pages.

Bar-Yossef et al. [1] and Henzinger et al. [9, 10] independently proposed to use random walks on the web to sample web pages. They present algorithms that in theory should lead to uniform random samples but cannot be implemented in their *pure* form. Instead, the implementations need to make some simplifications which lead to various biases in the resulting samples. Both evaluated their walks on different artificially generated graphs and on the web (at different times). Based on this work, Rusmevichientong et al. [13] proposed two different random walks, which in theory should lead to uniform random samples. One of their approaches can be implemented *without modifications*. However, they evaluated their approaches only on small artificially generated graphs consisting of 100,000 nodes. On these graphs they showed that their approaches and the approach in [1] lead to samples that reflect the indegree and outdegree distributions of the underlying graph correctly, while the approach by Henzinger et al. [9, 10] does not. Henzinger et al. [10] had found a bias in their approach for the indegree distribution but not for the outdegree distribution. More recently, Bar-Yossef et al. [2] showed how to generate a random sample of web pages relevant to a given user specified topic and Chakrabarti et al. [6] developed techniques to estimate the background topic distribution on the web. Both [2] and [6] use a variant of the sampling algorithm proposed in [1].

In the rest of the paper we will denote the algorithm proposed in [13] as *Algorithm A*, the algorithm proposed in [1] as *Algorithm B* and the algorithm in [10] as *Algorithm C*. Each algorithm consists of a *walk phase* that performs a random walk and of a *subsampling phase* that subsamples the web pages visited by the random walk. We performed the walk phase of each of these algorithms *on the web* with the *same* computation power and with the *same* amount of time. Then we experimented with different subsampling phases, including the ones proposed by the above papers. This resulted in four types of samples generated by Algorithm A, called *A Samples*, four types of samples generated by Algorithm B, called *B Samples*, and three types of samples generated by Algorithm C, called *C Samples*.

Our experiments provide the following new insights about the above mentioned algorithms: (1) A Samples and B Samples exhibit a strong bias to internally highly connected hosts with few outedges to other hosts. The reason is that Algorithm A and Algorithm B frequently had problems leaving such hosts. After a certain (large) number of consecutive visits of web pages on the same host, we say that the walk is *unable to leave* the host or, more informally, got *stuck* at a host. Both Algorithm A and Algorithm B have a problem with getting stuck. Algorithm C is designed to have a very low probability of getting stuck, due to *random resets*. Indeed in our experiments it was never unable to leave a host. (2) C Samples exhibit a bias towards high outdegree web pages. This was shown before for artificially generated graphs [13] but not for the web. Furthermore we show that C Samples show a bias towards high PageRank web pages. (3) We experimented with different subsampling phases for each algorithm. The subsampling techniques had an impact on A Samples and B Samples while they had only a very small impact on C Samples.

This paper is organized as follows: Section 1 describes the evaluated algorithms and their corresponding subsampling phases. Section 2 presents some challenges met during the

implementation and how we dealt with them. Section 3 presents the experiments and their results in detail. In Section 4 we give a comparison of results for sampling algorithms. We conclude with proposals for further work in Section 5.

1. Description of the algorithms

We define the *web graph* as a graph where every web page is a node and every hyperlink is a directed edge between the nodes. A *memoryless random walk* on the web graph is a Markovian chain that visits a sequence of nodes where the *transitions* between nodes depend only on the last node of the walk and not on earlier nodes. In a Markov chain on the web graph *states* correspond to web pages, i.e. the nodes on the web graph, and each visit to a node results in one *step* of the random walk. We call a step a *selfloop step* when the walk visits the same node in two consecutive steps of the walk by traversing a selfloop. We define the *visit count* of a node to be the number of visits to the node including selfloop steps. *Edges*, *degree*, *PageRank*, *inlinks*, *outlinks* and *selfloop* of a state are the edges, degree, PageRank value, inlinks, outlinks and selfloop of the corresponding node.

Each algorithm consists of two phases: (1) A *walk phase*, where a memoryless random walk is performed on the web graph. We denote the walk phase of Algorithm A, Algorithm B and Algorithm C as *Walk A*, *Walk B* and *Walk C* respectively. (2) The second phase is a *subsampling phase*, where either states or steps of the walk phase are subsampled randomly.

According to a fundamental theorem of Markovian chains, a random walk on an aperiodic and irreducible graph will converge to a unique stationary distribution. Once the walk reaches its unique stationary distribution, the probability of being in a node will not change although the walk takes more steps. Algorithm A and Algorithm B are designed to perform a random walk on an undirected, aperiodic and irreducible graph. On such a graph a random walk converges to a unique stationary distribution where the probability of being in a node is proportional to its degree. Walk A leads to a biased stationary distribution because the nodes do not have the same degree. If we subsample states after the point where the walk reaches the stationary distribution, high degree nodes will be more likely to be sampled. To remove this bias we subsample states or steps of Walk A with values inversely proportional to the corresponding node's degree. Walk B is performed on a regular graph, i.e. on a graph where each node has the same degree. Furthermore this regular graph has the above mentioned properties required for converging to a unique stationary distribution. Thus, in the stationary distribution values of Walk B every node is equally likely to be visited. Algorithm C is designed to perform a random walk on a directed, aperiodic and irreducible graph. This walk leads to a unique stationary distribution where the probability of being in a node is equal to its PageRank value. In other words Walk C has a biased stationary distribution, as does Walk A. To get a uniform random sample of Walk C, we subsample its *states* with values inversely proportional to the PageRank values of the corresponding nodes. We next describe the algorithms in more detail.

1.1. Algorithm A

Walk phase: Consider the following random walk on an *undirected* graph. From the current node choose an adjacent edge uniformly at random and select the other endpoint

of the edge as next node to visit. It can be proven that if run long enough on an *undirected, irreducible and aperiodic* graph this random walk converges to a unique stationary distribution where the probability of visiting a node is proportional to its degree.

Algorithm A executes Walk A on the web graph which it modifies as follows: (1) It gives a selfloop to each node that does not yet have a selfloop to make the web graph aperiodic. (2) It ignores the direction of the (directed) hyperlinks. The latter leads to complications in the implementation since the inlinks of a node in the web graph can not be determined directly from the corresponding web page. Additionally the web graph changes constantly as web pages are edited. We deal with the former problem by querying a web search engine and retrieving up to 10 inlinks per node, chosen randomly from all returned inlinks. We do not retrieve more inlinks since it was shown experimentally in [1] that Algorithm A returns better results when the number of retrieved inlinks is limited. These inlinks and outlinks together with the inlinks (if any) from previously visited nodes form the set of adjacent edges of a node. To deal with the changes in the web we store the set of adjacent edges of a node at the first visit of the node in a database. At every later visit of the node the set of its adjacent edges is taken from the database. This guarantees that the degree of a node does not change during the execution of the walk.

Subsampling phase: After Walk A reaches its unique stationary distribution each node can be the next step of the walk with probability proportional to its degree. To remove this bias, states or steps are subsampled randomly with probability inversely proportional to their degree after the step where the walk reached the stationary distribution. We wanted to implement the algorithms described in [1] and [13] as closely as possible, however it was not clearly described whether they subsampled states or steps. Thus we created two types of samples, one subsampling states and one subsampling steps. The number of steps until the walk has reached a stationary distribution is called the *mixing time.* No bounds for the mixing time on the web graph are known. However, intuitively the distribution of states towards the end of the walk should lead to better results than the distribution of all the states in the whole walk. We tested this intuition by exploring the following different subsampling phases. (1) We determined all the states visited in the last half of the steps of Walk A and subsampled them randomly with probability inversely proportional to their degree. This sample is called *A_StatesOnLastHalf.* (2) We determined all states visited in the last quarter of the steps of Walk A and subsampled them randomly with probability inversely proportional to their degree. This sample is called *A_StatesOnLastQuarter.* (3) From the last half of the steps of Walk A, the steps are subsampled randomly with probability inversely proportional to the corresponding web page's degree. This sample is called *A_StepsOnLastHalf.* (4) From the last quarter of the steps of Walk A, the steps are subsampled randomly with probability inversely proportional to the corresponding web page's degree. This sample is called *A_StepsOnLastQuarter.*

1.2. Algorithm B

Walk phase: Consider the same random walk as for Walk A on an *undirected, regular and irreducible* graph. If run long enough this random walk converges to a uniform distribution of the nodes. The web graph is neither undirected nor regular. The web graph is modified as described in Section 1.1 to make it undirected. To make it regular we add enough selfloops to each node to increase their degree to max. Following [1] we set $max = 10,000,000$. Thus, the only difference between Walk A and Walk B is the number of selfloops in the graph.

Subsampling phase: We subsampled Walk B in the same four ways as Walk A, but subsampling states uniformly at random creating B_StatesOnLastHalf, B_StatesOnLastQuarter and subsampling steps uniformly at random creating the samples B_StatesOnLastHalf, B_StepsOnLastQuarter.

1.3. Algorithm C

Walk phase: Algorithm C tries to imitate the PageRank random walk [4] as closely as possible. When choosing the next node to visit, Walk C first flips a biased coin. With probability $d = 1/7$ it performs a *random jump* or *random reset*, described below. With probability $1 - d$, it chooses an outlink of the current node uniformly at random and selects the head of the selected outlink as next step of the walk. We say that the algorithm *traverses* the chosen outlink. If a chosen node does not have any outlinks or if it cannot be fetched, a random jump is performed. Ideally a random jump would jump to a randomly selected node of the web graph. However, the walk does not know all the nodes on the web graph. Instead it can choose a node out of all *visited* or all *seen* nodes. A node is *seen* if it either has already been visited or if it is the head of an outlink of a visited node. However, even when restricting the random jumps to all visited or all seen nodes, there is a potential problem. As pointed out in [9] if almost all of the seen nodes are on the same host, a random jump would with high probability jump to a node on this host. As a result it is possible that the walk gets stuck on this host. To remedy this problem [9] proposed to perform a random jump in the following biased way: First select a host from all the visited hosts uniformly at random, then select a web page from all the visited web pages on that host uniformly at random and finally visit the node corresponding to the selected web page. In our implementation we "got stuck" in domains using this approach and thus we added one additional layer, the domain[2] layer. Additionally we switched from *visited* to *seen* entities. A *seen host* is a host on which the walk has seen a web page and a *seen domain* is a domain on which the walk has seen a web page. Our Walk C first selects a domain uniformly at random from all the seen domains, secondly it selects a host uniformly at random from all the seen hosts in that domain, then it selects a web page uniformly at random from all the seen web pages in that host and finally the walk visits the node corresponding to the selected web page. Since the set of seen nodes is on the average a factor of roughly 10 larger than the set of visited nodes, this modification allowed us to more closely imitate the PageRank random walk that chooses a random node out of *all* nodes on the web graph in random jump phase. Due to our way of imitating PageRank random walk our Walk C is not memoryless since it keeps track of all the visited states as well as their outlinks.

Subsampling phase: Following [10] we use three different subsampling phases to subsample states of Walk C. One is simply a uniform random sample of all the nodes, called *C_Random*. However this will be biased towards high PageRank nodes as they are more likely to be visited. The other two sampling techniques try to correct for this bias. The idea is to sample inversely proportional to PageRank values. Since the PageRank values for the whole web is not known, a *PageRank substitute* is used during the sampling. It is computed in one of the two possible ways: (1) The PageRank of the subgraph of the visited states is computed and the visited states are subsampled inversely proportional to their PageRank values. This sample is called *C_PR*. (2) The ratio of the number of visits of a node to the total number of steps of the walk is called as *visit ratio* of the node. The PageRank random

[2]We denote by *domain* second level domains like *epfl.ch* or *berkeley.edu*.

walk converges to a unique stationary distribution where the probability that a node is visited is proportional to its PageRank value. In the limit, i.e. when the length of the walk goes to infinity, the visit ratio values of the nodes are equivalent to PageRank values of the nodes. For the *C_VR* sample the states are sampled with probability inversely proportional to the visit ratio values of the nodes corresponding to them.

2. Implementation details

In this section we describe various complications that arose during the implementation and how we addressed them.

Fetching: In our walks we did not crawl the web pages whose encoded version were more than 300 characters long following [1]. We only downloaded HTML/Text documents, ignored Javascript links and frame src links on them. To avoid wasting bandwidth, we downloaded only the first 5 MB of a web page. We stopped fetching a web page if we could not download it after $1,500$ seconds. In this case Walk A and Walk B selected uniformly at random a sibling of the current node, while Walk C made a random reset.

Host overload: If a walk tried to fetch web pages on the same host consecutively more than $3,000$ times, we put the walk to sleep for 20 minutes to avoid host overload. If this happened 12 times on the same host, we stopped the walk and declared that it was unable to leave the host.

Parallel links: If there were multiple parallel hyperlinks from one web page to another, we kept only two of them.

HTTP and HTML redirects: If a web page redirected to an another web page we treated them as the same node in the web graph. This applied iteratively to the whole "redirect chain". We combined the inlinks of all the web pages in the redirect chain. If this combination resulted in more than 10 inlinks retrieved from a search engine, we stored only a uniform random sample of 10 of them. If a newly visited web page redirected to a previously visited web page, we did not retrieve inlinks for the new web page and instead used the inlinks of the previous web page. We followed only up to 10 HTTP or HTML redirects. If there were more than 10 redirects or we detected a redirection loop, Walk A and Walk B selected a random sibling of the previous node while Walk C made a random reset.

Truncation: URLs with and without session id usually represent the same web page. Thus, we treated them as one node to avoid bias during the walk and the subsampling phase. Session ids are notoriously hard to detect in general, but frequently they come after question marks in the URLs of the web pages. Thus we truncated URLs with question mark at the question mark, but only under certain conditions. First we experimented with a walk that always truncated at session ids. However, sites for webmaster referral programs frequently encode a web page after the question mark and redirect to it. Truncating after the question mark prevented the walk to follow those redirections. Truncating only if no error page is returned does not solve the problem either because the truncated page might not return an error page but cause a new redirection. Thus we chose the following strategy: When fetching a web page the walk first follows all redirects that it can follow and if the URL of the final web page in the redirect chain contains a question mark it is truncated. If the truncation leads to an error page or a new HTTP redirect, the walk undoes the truncation.

Speed up: To speed up the walk phase of all the algorithms we used multiple walks in parallel which shared the database. These walks started from the same initial node and they were not completely independent of each other since they shared the database. However, the shared database only makes sure that all the walks "see the same graph", i.e., that the edges adjacent to a node remain same throughout all the walks.

Sampling steps or states: In the subsampling phase the last half of the steps of the multiple walks are merged and a subgraph is formed from these steps. We recorded the number of times the merged walk spent at each node, namely the *visit count* for each node. For the A_StatesOnLastHalf sample the states of the merged Walk A on the formed subgraph are sampled with probability inversely proportional to the degree of the states. For the A_StepsOnLastHalf sample the states of the formed subgraph are sampled with probability proportional to the state's visit count divided by its degree. The samples from the last quarter of the steps of the multiple walks are taken exactly the same way except that we formed the subgraph from the last quarter of the steps of each walk. We proceded in the same way for the other algorithms. We set the sampling probabilities such that each sample consisted of around $10,000$ nodes.

Average of samples: For each sample type of each algorithm we took 5 samples. Each number given in Section 3 is actually the average of these 5 samples.

3. Experiments

Recall that Walk A and Walk B differ only in the number of selfloops in the underlying graphs on which they are performed. To save resources we did not perform a random walk for Algorithm A and a random walk for Algorithm B. Instead we performed only one random walk ignoring selfloop steps for Algorithm A and Algorithm B. We call this *Walk AB*. In a postprocessing step we simulated Walk A and Walk B with selfloop steps by flipping a suitably-biased random coin (dependent on the algorithm) once at every step of Walk AB and adding a suitable number of selfloop steps when the coin comes up heads. For Walk B the probability of traversing a selfloop is very high. Thus instead of flipping often a random coin each deciding on just one step, namely the next one, we model the number of selfloop steps at the current node by a geometric random variable and determine how many selfloop steps are executed at the current node using one random number. This approach was already proposed by [1]. It results in exactly the same random walk as Walk A, resp. Walk B, would have performed with the same coin flips and random walk choices. As a result of simulating Walk A and Walk B from one common walk, the data for Walk A and Walk B are highly correlated. However this has the positive side-effect that it allows to evaluate whether Algorithm A, which is a modification of Algorithm B, does lead to better results, as claimed by [13]. In our implementation Algorithm A and Algorithm B agree in all non-selfloop transitions. Thus, if changing the number of selfloops per state and subsampling states inversely proportionally to degree instead of randomly does indeed change the quality of the sample as claimed by [13] our evaluation should show that. We performed a completely separate random walk for Algorithm C.

We ran both walks, Walk AB and Walk C, for 240 hours on two identical machines equipped with a Intel Pentium 4 processor 3.0 Hz (HyperThreading enabled), 4 GB of RAM, and a 4 Seagate HD (250GB each) in RAID5 on 3ware RAID controller 8506. As database we used PostgreSQL 7.4.8. The implementations shared as much code as possible. Both Walk AB and Walks C started from *http://www.yahoo.com/* and used 50 walks in parallel

as explained in Section 2. Three of the walks of Walk AB had to be stopped because of host overloading before the end of the walk. We removed their nodes and transitions from Walk AB. None of the C walks had to be stopped. Walk AB visited 842,685 nodes, leading to 1.7 million steps for Algorithm A and 4.3 trillion steps for Algorithm B. Walk C visited 695,458 nodes with almost 1 million steps.

Random walk	Duration	# of visited nodes	# of seen hosts	# of seen domains
AB	240 hours	842,685	2,360,329	1,041,903
C	240 hours	695,458	1,814,444	991,687

Table 1: Random walks on the web

Table 1 shows that the number of seen domains is almost identical for Walk AB and Walk C. When compared to Walk AB, Walk C visited 20% fewer nodes and saw about 25% fewer hosts. This drop is not surprising since Walk C made a random jump to an already seen node in about 21% of the transitions while Walk AB does not perform random jumps.

In Walk AB about 58% of the non-selfloop transitions traversed an outlink, 42% traversed an inlink. We conjecture that the reason for this imbalance is that we artificially limit the number of inlinks at 10, while the average number of outlinks for Walk AB is 46.71.

In Walk C an outlink was traversed in 79% and a random jump happened in 21% of all the transitions. This number does not vary much over the length of the walk. Based on the reset probability of 1/7 one would expect that random jumps account only in 14% of the transitions in Walk C. However dead ends, problems while fetching a page, long redirect chains, or redirect loops all caused a random jump and are the reason for the additional 7% of transitions with random jumps.

Each of the following subsections compares Walk A, Walk B, Walk C and the samples generated by them using different measures. The first subsection compares the algorithms using their "nodes per host" distribution. The following subsections compare the algorithms using their "PageRank bias" and "outdegree" distribution. These subsections all point out the weaknesses of different sampling approaches. The last two subsection presents results for connectivity-independent statistics namely the "top level domain" and the "document content length" distribution.

3.1. Nodes per host distribution

A uniform random sample of the nodes on the web graph should contain about as many different hosts as there are nodes in the sample [3]. This is the case for each of C Samples, each contain about 9,500 unique hosts out of about 10,000 nodes. However, A Samples and B Samples contain many fewer hosts even though we omitted all the data from the three walks of Walk AB that were stopped because they were unable to leave a host.

As can be seen in Table 2, all A Samples and B Samples except the B samples subsampling states contain about three times as many visited nodes on the host *fr.shopping.com* than from other hosts. This is a significant bias towards the nodes on that host being due to multiple walks almost "getting stuck" in it. It seems like a fundamental flaw in Algorithm A and Algorithm B: They have a large problem with hosts that are highly connected within but have few edges leaving them. Here is an intuitive explanation: Consider an undirected graph of n nodes, consisting of a complete graph of $n/2$ nodes with a chain of $n/2$ nodes attached to one of the nodes in the complete graph. If Walk A were run on this graph, it

# of nodes	% of nodes	Host
A_StatesOnLastHalf		
2051	20.60%	fr.shopping.com
874	8.78%	www.rechtschutzversicherung.de
520	5.22%	www.friday.littledusty.org
A_StatesOnLastQuarter		
1850	18.94%	fr.shopping.com
648	6.63%	www.rechtschutzversicherung.de
627	6.41%	classifieds.fr
A_StepsOnLastHalf		
2849	29.49%	fr.shopping.com
874	9.04%	www.hostpooling.com
771	7.98%	www.friday.littledusty.org
A_StepsOnLastQuarter		
3170	32.73%	fr.shopping.com
825	8.44%	www.hostpooling.com
677	6.92	www.friday.littledusty.org
B_StatesOnLastHalf		
916	9.11%	fr.shopping.com
455	4.53%	www.rechtschutzversicherung.de
356	3.54%	www.smart.com
B_StatesOnLastQuarter		
730	7.34%	fr.shopping.com
302	3.03%	www.rechtschutzversicherung.de
256	2.57%	www.smart.com
B_StepsOnLastHalf		
2551	26.63%	fr.shopping.com
880	9.18%	www.hostpooling.com
521	5.44%	www.friday.littledusty.org
B_StepsOnLastQuarter		
2787	29.13%	fr.shopping.com
833	8.70%	www.hostpooling.com
542	5.67%	classifieds.fr

Table 2: The hosts with the most nodes in A Samples and B Samples

would have a very good chance of getting stuck in the complete subgraph when run long enough. To avoid this problem Algorithm B added selfloops to make the graph regular. As a result the walk is equally likely to "get stuck" on the chain as in the complete subgraph. However, the fundamental problem of "getting stuck", i.e., staying within a small part of the graph, is not solved. Walk C avoids this problem by performing random jumps. Indeed, the host with the largest number of states in any of C Samples, *www.amazon.com*, has only 32 nodes in the sample.

Table 2 shows that subsampling from the last half of the steps or the last quarter of the steps does not seem to have a impact on the resulting samples for Algorithm A and Algorithm B. The top 2 hosts with the most nodes are same in A Samples and B Samples subsampling states. Subsampling from the last half of the steps or the last quarter of the

Sample	# of unique hosts
A_StatesOnLastHalf	1,449
A_StatesOnLastQuarter	1,277
A_StepsOnLastHalf	671
A_StepsOnLastQuarter	702
B_StatesOnLastHalf	3,405
B_StatesOnLastQuarter	3,442
B_StepsOnLastHalf	656
B_StepsOnLastQuarter	750
C_VR	9,498
C_PR	9,504
C_Random	9,499

Table 3: The number of unique hosts in A Samples, B Samples and C Samples

steps does not affect the top 3 hosts list for A Samples and B Samples subsampling steps either. On the other hand subsampling states or steps seems to make a difference. With no exception *fr.shopping.com* is the host with the most nodes in A Samples and B Samples. However in the B_StatesOnLastHalf sample and in the B_StatesOnLastQuarter sample the percentage of nodes on the top hosts is smaller when compared to the other A Samples and B Samples. Table 3 presents the number of unique hosts in A Samples, B Samples and C Samples. It shows that the number of unique hosts in B Samples subsampling states is almost 5 times greater than the number for B Samples subsampling steps. For C Samples as can be observed in Table 2 the number of unique hosts is roughly same as the number of the nodes. Thus we can conclude that sampling states leads to less biased distribution for the number of nodes per host.

3.2. PageRank bias

Page-based analysis: Walk C tries to visit nodes roughly according to their PageRank values. Thus the most frequently visited nodes should have high PageRank values. Table 4 presents the top visited 10 nodes[3] during Walk C. We also give the PageRank as returned by the Google toolbar next to each node. For one node no PageRank is returned, all others have Toolbar PageRank 7 or above. We conclude that our walk did indeed succeed in visiting high PageRank nodes more frequently than other nodes. We observed no such bias towards high PageRank nodes in Walk AB as can be seen from Table 5. Indeed no PageRank value is returned by the Google toolbar.

We call the PageRank of the subgraph traversed in Walk C the *subgraph PageRank*[4]. Figure 1 shows the percentage of nodes in certain subgraph PageRank ranges for the whole crawled subgraph, for the C_PR sample and for the C_VR sample. Since the C_PR sample was created by subsampling states inversely proportional to the subgraph PageRank values we would expect that nodes with low subgraph PageRank values are more frequent in the

[3]The most visited node is a web page on a tracking site for website visitors. This web page is the result of our truncation of URLs after questions marks for many different web pages, i.e., it is an artifact of our implementation of session id handling.

[4]The subgraph PageRank value of a state can be very different from its PageRank value in the whole web graph.

PRank	Visit count	Node
8	929	http://extreme-dm.com/tracking/
10	810	http://www.google.com/
8	696	http://www.macromedia.com/shockwave/download/download.cgi
-	478	http://www.sitemeter.com/default.asp
10	364	http://www.statcounter.com/
7	336	http://www.mapquest.com/features/main.adp
10	312	http://www.microsoft.com/windows/ie/default.mspx
9	312	http://www.yahoo.com/
10	294	http://www.adobe.com/products/acrobat/readstep2.html
9	286	http://www.blogger.com/start

Table 4: The most visited 10 nodes of our Walk C

PRank	Visit count	Node
-	10,228	http://66.40.10.184/browses/AlphaBrowses/NF_manufacturer.asp
-	7,496	http://www.mix-networks.com/forum/index.php
-	7,436	http://www.fatmp3.com/sitemap.html
-	6,899	http://bbs.dingding.org/RssFeed.asp
-	5,005	http://www.hotels55.info/a-z-test.php
-	2,457	http://www.hostpooling.com/berlin/hotel/billig/lease/home_&_garden.htm
-	2,434	http://sms.3721.com/rsearch/ivr.htm
-	2,411	http://www.sh-netsail.com/www7/default.asp
-	2,185	http://www.hostpooling.com/berlin/hotel/billig/lease/health.htm
-	1,999	http://forums.gamedaily.com/index.php

Table 5: The most visited 10 nodes of Walk AB

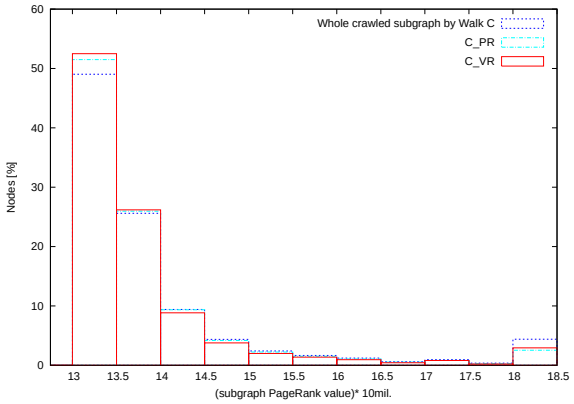


Figure 1: PageRank value distribution in the crawled subgraph for Walk C, in C_PR sample and in C_VR sample

sample than in the graph as a whole and very few nodes with high subgraph PageRank values are in the sample. This is exactly what we see in Figure 1. We also included the C_VR sample in the figure although we did not use PageRank values for getting the C_VR sample. In Figure 1 we see that the C_VR sample behaves very similar to the C_PR Sample.

This shows that using the visit ratio as a substitute to PageRank works as well. However neither subsampling phase is powerful enough to erase the PageRank bias present in Walk C.

Host-based analysis: The visit count of a node is the number of visits to the node as defined in Section 1. The visit count of a host is the sum of the visit counts of the nodes, namely web pages, on that host. Table 6 shows the top visited hosts of Walk C together with their visit counts. It shows a clear bias towards well-known popular hosts. The most visited 10 hosts of Walk AB show no obvious bias to well-known, popular hosts. Table 6 shows also the top visited hosts of Walk 2 in [9]. Only three of the hosts, namely Amazon, Microsoft and Adobe, are in the top 10 list for both years. We attribute these differences to the big changes that have occurred in the web in the mean time.

Our Walk C		Walk 2 in [9]	
Visit count	Host	Host	Visit count
4,509	www.macromedia.com	www.microsoft.com	32,452
3,262	www.amazon.com	home.netscape.com	23,329
2,848	www.google.com	www.adobe.com	10,884
2,246	www.microsoft.com	www.amazon.com	10,146
1,617	www.cyberpatrol.com	www.netscape.com	4,862
1,462	www.sedo.com	excite.netscape.com	4,714
1,412	www.adobe.com	www.real.com	4,494
1,132	www.cafepress.com	www.lycos.com	4,448
1,069	www.blogger.com	www.zdnet.com	4,038
929	extreme-dm.com	www.linkexchange.com	3,738

Table 6: The most visited 10 hosts of our Walk C and of Walk 2 in [9]

3.3. Outdegree distribution

As was shown in the literature the outdegree distribution of the nodes on the web graph follows a power law. Thus the outdegree distribution of a uniform random sample of the nodes on the web graph should ideally follow a power law distribution. In Figure 2(a), Figure 2(b) and Figure 2(c) we present the outdegree distribution on log-log scale for A Samples, B Samples and C Samples respectively. In these figures for all the samples we observe that the percentage of nodes with high outdegree is lower when compared to the percentage of nodes with low outdegree.

The power law exponent for outdegree distribution of the nodes on web graph is given as 2.72 in [5]. It would be interesting to see how our samples agree with this value. For the A_StatesOnLastHalf sample and the B_StepsOnLastHalf sample the power law exponent is 2.01. On the other hand for the B_StatesOnLastHalf sample the exponent is 1.41. In other words, B samples sampling states are more biased to high outdegree nodes when compared to other B Samples and A Samples. The outdegree power law exponent is about 1.49 for C Samples. This indicates that C Samples have a bias to high outdegree nodes. For none of our samples does the power law exponent agree with the value in the literature, giving evidence that all our samples are biased to high outdegree nodes.

The average outdegree on the web graph was estimated by prior work [11] to be around 10. A uniform random sample of the web graph should have this property. In order to investigate this we present the statistics about outdegree distribution of A Samples,

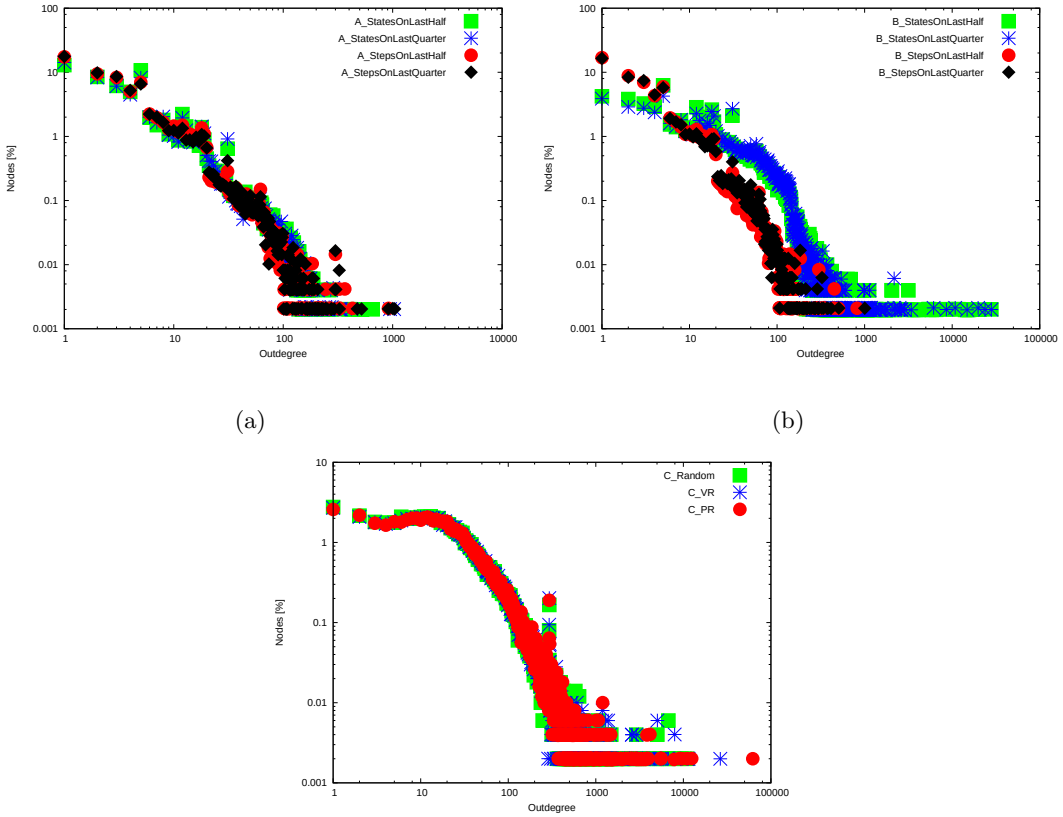


Figure 2: Outdegree distribution of nodes in (a) A Samples (on log-log scale), (b) B Samples (on log-log scale), (c) C Samples (on log-log scale)

B Samples and C Samples in Table 7. As can be seen in this table all A Samples, the B_StepsOnLastHalf sample and the B_StepsOnLastQuarter sample have an average outdegree of roughly 10. However, the C Samples, the B_StatesOnLastHalf sample and the B_StatesOnLastQuarter sample have an average outdegree that is a factor of 4 to 6 larger. We already discussed above that the C Samples have a bias to high outdegree nodes. At a first glance looking at the average outdegree seems to indicate that the A and the B samples sampling steps have no bias towards high outdegree nodes, contradicting our above finding based on the power law exponent. However a closer investigation showed that many of the most frequently visited hosts have nodes with outdegree 0. Thus the very biased distribution of nodes per host of the samples sampling steps leads to their low average outdegree.

Sample	Outdegree	
	Avg	Max
A_StatesOnLastHalf	8.03	656
A_StatesOnLastQuarter	7.80	1,031
A_StepsOnLastHalf	7.01	916
A_StepsOnLastQuarter	6.95	1,041
B_StatesOnLastHalf	46.76	27,994
B_StatesOnLastQuarter	47.08	27,994
B_StepsOnLastHalf	6.63	822
B_StepsOnLastQuarter	6.63	1,003
C_VR	59.06	26,423
C_PR	60.22	62,021
C_Random	57.82	11,138

Table 7: Statistics about outdegree distribution

3.4. Top level domain (TLD) distribution

A top level domain is the last part of the domain name, like ".com" or ".net". The distribution of web pages over the top level domains is not known, but could be estimated if we could sample the web uniformly at random. Even though, unlike for the outdegree distribution or the nodes per host distribution we do not know the "correct" answer, it is interesting to compare the results achieved by the different sampling techniques. A rough agreement would give us an indication of what the correct answer is likely to be. Thus in this subsection we present the top level domain distribution for A Samples, B Samples and C Samples (see Table 8 and Table 9).

TLD	A Samples				B Samples			
	A_States OnLast Half	A_States OnLast Quarter	A_Steps OnLast Half	A_Steps OnLast Quarter	B_States OnLast Half	B_States OnLast Quarter	B_Steps OnLast Half	B_Steps OnLast Quarter
.com	53.81	50.82	64.87	62.77	49.29	44.26	64.55	62.67
.edu	0.22	0.24	0.06	0.08	0.41	0.32	0.05	0.08
.org	8.26	8.67	12.57	11.72	4.18	4.23	10.89	8.70
.net	6.83	7.39	8.42	7.71	8.93	10.11	8.17	7.85
.jp	0.81	0.96	0.25	0.38	2.09	2.62	0.24	0.39
.gov	0.13	0.15	0.05	0.03	0.24	0.20	0.05	0.04
.uk	1.26	0.66	0.51	0.29	1.40	0.99	0.59	0.33
.us	0.13	0.16	1.02	1.29	0.39	0.46	0.97	1.87
.de	11.96	11.02	3.70	3.92	7.74	6.03	3.49	3.21
.ca	0.16	0.12	0.04	0.06	0.33	0.26	0.04	0.05
.fr	5.68	8.44	2.32	4.36	1.53	2.00	3.56	7.10

Table 8: Top level domain distribution for A Samples and B Samples

Recall that Walk AB and Walk C were performed completely independent of each other. Still the samples generated from them roughly agree: About 44-65% of the nodes, namely web pages, are in ".com" domain, making it clearly the largest domain on the web. The domains ".net " and ".org" contain about 4-9% of the nodes.

TLD	C Samples			Samples from 2000	
	C_Random	C_PR	C_VR	B Sample from [1]	C_VR from [10]
.com	63.20	62.94	63.13	49.15	45.62
.edu	0.64	0.60	0.67	8.28	9.84
.org	9.79	9.94	9.82	6.55	9.12
.net	6.19	6.14	6.20	5.60	4.74
.jp	0.44	0.48	0.46	2.87	3.87
.gov	0.47	0.46	0.49	2.08	3.42
.uk	3.28	3.34	3.26	2.75	2.59
.us	0.63	0.62	0.56	1.12	1.77
.de	3.28	3.32	3.28	3.67	3.26
.ca	0.83	0.83	0.84	1.58	2.05
.fr	0.43	0.40	0.43	1.01	0.99

Table 9: Top level domain distribution for C Samples, B Sample from [1] and C_VR sample from [10]

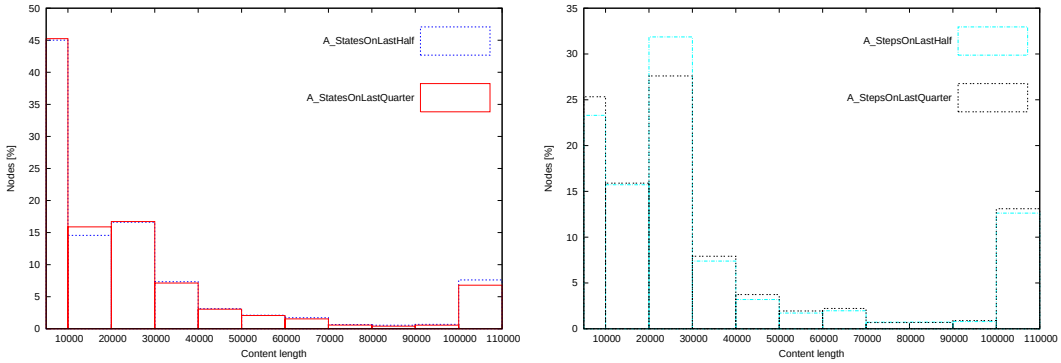
The domains ".de" and ".fr" show large variances in the percentage of the nodes in them. For ".de" the large values (around 11%) for the A_StatesOnLastHalf sample and the A_StatesOnLastQuarter sample are due to the high frequency of a German host, which in turn is caused by the inability of Walk AB of leaving highly connected hosts. Thus these percentages are artificially high and should be ignored. Additionally all percentages for the ".de" domain are inflated due to the fact that we performed our walks from Switzerland for which the country of originator for domain forwarding is Germany.

The results for top level domain distribution from [1] and from [10] (Table 9) roughly agree and ".com" is the largest top level domain as in our A Samples, B Samples and C Samples.

3.5. Document content length distribution

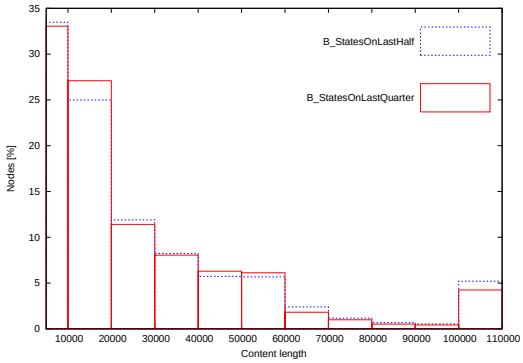
In this subsection we study the document content length distribution for A Samples, B Samples and C Samples. We bucketed the content length values as follows: the first bucket, 0-10k, contains the percentage of nodes (web pages) in the samples whose content length is between 0 and 10k. The definition for the other buckets is analogous. For the last bucket (100-110k) we put all the nodes whose content length is greater than 100k, causing a relatively large value in that bucket for all the samples.

Figure 3 presents the document content length distribution for the different samples. B Samples subsampling states (Figure 3(c)) have a similar document content length distribution as C Samples (Figure 3(e)). Generally the percentage of nodes per bucket is monotonically decreasing with the content length. However, there is a spike for A samples sampling states in bucket 0-10k and a spike for A samples and B samples sampling steps in bucket 20-30k. A detailed analysis showed that these spikes are caused by the uneven distribution of nodes over hosts.

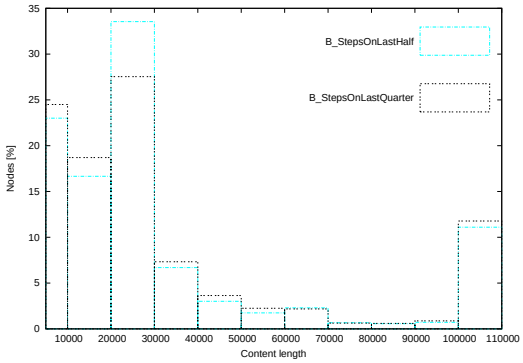


(a)

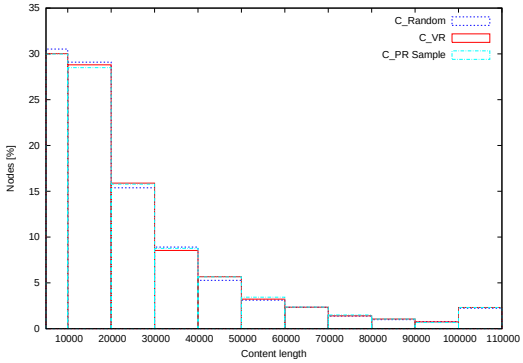
(b)



(c)



(d)



(e)

Figure 3: Content length (in bytes) distribution of nodes (web pages) in (a) A Samples subsampling states, (b) A Samples subsampling steps, (c) B Samples subsampling states and (d) B Samples subsampling steps (e) C Samples

4. Comparison of techniques

In this section we compare the different samples of each algorithm over all the different measures we have used.

Subsampling from the last half or from the last quarter of the steps: Since we ran walks for a fixed amount of time starting from the same node, the results are somewhat influenced by the chosen starting node. The longer the walks run, the smaller this bias should become. Thus we wanted to evaluate whether subsampling Walk A and Walk B from the last quarter of the steps gives improved results over subsampling these walks from the last half of the steps. Our results indicate that this is not the case. For none of the samples did we see a large difference in the results whether they were based on the last half or the last quarter of the steps. Thus either approach seems to work equally well and the starting-node bias seems small.

Subsampling from steps versus from states: After determining the set of steps to subsample from, we either subsampled steps directly from these steps or we determined the states represented by them and subsample the states. Obviously, when a random walk was unable to leave a host for a long time and frequently revisits nodes on the same host, these nodes have a higher chance of being in the sample when we subsample steps than when we subsample states. This can be seen in Table 2: When steps are subsampled, a much larger percentage of the samples belongs to the same host than when states are subsampled. As a result various measures exhibit unexpected spikes for the samples based on steps, see for example the document content length distribution in Figure 3. This indicates that it is better to subsample states. However, for top-level domain distribution the samples based on steps both for Walk A and for Walk B showed a large agreement with C Samples, while the samples based on states disagreed with each other and with the C Samples. Further investigation is necessary to understand this behaviour.

Algorithm A versus Algorithm B versus Algorithm C: Algorithm C has a clear bias towards high PageRank and high outdegree nodes. However, it generates a roughly uniform distribution of nodes per host. Algorithm A and Algorithm B generate a very unbalanced distribution of nodes over hosts, with more than 30% of the nodes in the sample belonging to only three hosts. As a consequence it is hard to believe that the results produced by this sample are representative of the whole web. All A Samples as well as B Samples subsampling steps exhibit this problem. Thus Algorithm B combined with state subsampling appears superior to Algorithm A. Recall that Algorithm A and Algorithm B were both implemented by the same walk. They differ however, (1) by the number of selfloops of the nodes and (2) by the subsampling probabilities (inversely proportionally to the degree for Algorithm A and uniformly at random for Algorithm B). Let us compare the A Samples subsampling states with the B Samples subsampling states. Both subsample from the states in the last half or in the last quarter of the steps of the walk. There are two possible reasons for the different quality of their samples: (1) Due to the selfloops the set of nodes from which Algorithm A and Algorithm B sample is very different. (2) Due to the probabilities used for subsampling different nodes are picked. To determine which of these reasons applies we compared the set of nodes used to subsample from. Our analysis showed they are almost identical for Algorithm A and Algorithm B. Thus, the subsampling probabilities are the reason for the difference in host frequency distribution for A Samples and B Samples subsampling states.

5. Conclusions and future work

We compared Algorithm A, Algorithm B and Algorithm C under conditions that are as equal as possible. Walk C has a clear bias towards high PageRank and high outdegree web pages and there seems to be no obvious way of correcting it. Algorithm A and Algorithm B has a serious problem with "getting stuck" in hosts. This had a clear impact on the nodes per host, outdegree, top level domain and document content length distribution. However, we believe that this problem can be corrected. We tried to eliminate the problem by stopping the walk when it could not leave a host for a large number of steps. However, a better approach might be to perform a random reset every x steps, like in Algorithm C. This is also the approach taken by [2] and by [6] in their work on the distribution of topics on the web.

References

[1] Z. Bar-Yossef, A. C. Berg, S. Chien, J. Fakcharoenphol, and D. Weitz. Approximating aggregate queries about web pages via random walks. In *International Conference on Very Large Databases (VLDB)*, pages 535–544, 2000.

[2] Z. Bar-Yossef, T. Kanungo, and R. Krauthgamer. Focused sampling: Computing topical web statistics. Technical report, IBM T.J Watson Research Center, 2005.

[3] K. Bharat, B. Chang, M. R. Henzinger, and M. Ruhl. Who links to whom: Mining linkage between web sites. In *International Conference on Data Mining (ICDM)*, pages 51–58, 2001.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems*, pages 107–117, 1998.

[5] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. In *International World Wide Web Conference (WWW)*, volume 33, pages 309–320, 2000.

[6] S. Chakrabarti, M. M. Joshi, K. Punera, and D. M. Pennock. The structure of broad topics on the web. In *International World Wide Web Conference (WWW)*, pages 251–262, 2002.

[7] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *International Conference on Very Large Databases (VLDB)*, pages 200–209, 2000.

[8] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of web pages. In *International World Wide Web Conference (WWW)*, pages 669–678, 2003.

[9] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. Measuring index quality using random walks on the web. In *International World Wide Web Conference (WWW)*, pages 213–225, 1999.

[10] M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform url sampling. In *International World Wide Web Conference (WWW)*, pages 295–308, 2000.

[11] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–17, 1999.

[12] S. Lawrence and C. L. Giles. Accessibility of information on the web. In *Nature*, volume 400, 1999.

[13] P. Rusmevichientong, D. M. Pennock, S. Lawrence, and C. L. Giles. Methods for sampling pages uniformly from the world wide web. In *AAAI Fall Symposium on Using Uncertainty Within Computation*, pages 121–128, 2001.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 31–50
www.stacs-conf.org

PROFINITE METHODS IN AUTOMATA THEORY

JEAN-ÉRIC PIN [1]

[1] LIAFA, Université Paris-Diderot and CNRS, Case 7014, 75205 Paris Cedex 13, France.
E-mail address: Jean-Eric.Pin@liafa.jussieu.fr

Abstract. This survey paper presents the success story of the topological approach to automata theory. It is based on profinite topologies, which are built from finite topogical spaces. The survey includes several concrete applications to automata theory.

In mathematics, p-adic analysis is a powerful tool of number theory. The p-adic topology is the emblematic example of a *profinite topology*, a topology that is in a certain sense built from finite topological spaces. The aim of this survey is to convince the reader that profinite topologies also play a key role in automata theory, confirming once again the following quote of Marshall Stone [38, p.814]:

> 'A cardinal principle of modern mathematical research may be stated as a maxim: "One must always topologize" '.

Unfortunately, this topic is rather abstract and not really intuitive. In particular, the appropriate framework to present the whole theory, namely *uniform spaces*, is unlikely to be sufficiently familiar to the average participant to STACS. To thwart this "user unfriendly" aspect, I downgraded from uniform spaces to metric spaces in this survey. This is sufficient to address most of the theory and it certainly makes the presentation easier to follow. When uniform spaces are really needed, I simply include a short warning addressed to the more advanced readers, preceded by the sign  . More details can be found in specialized articles [1, 2, 3, 5, 27, 30, 40].

Profinite topologies for free groups were explored by M. Hall in [13]. However, the idea of profinite topologies goes back at least to Birkhoff [8, Section 13]. In this paper, Birkhoff introduces topologies defined by congruences on abstract algebras and states that, if each congruence has finite index, then the completion of the topological algebra is compact. Further, he explicitly mentions three examples: p-adic numbers, Stone's duality of Boolean algebras and topologization of free groups. The duality between Boolean algebras and Stone spaces also appears in [1], [2, Theorem 3.6.1] and [31]. It is also the main ingredient in [12], where the extended duality between lattices and Priestley spaces is used. This duality approach is so important that it would deserve a survey article on its own. But due to the lack of space, I forwent, with some regrets, from presenting it in the present paper. The interested reader will find duality proofs of the results of Sections 4 and 5 in [12].

1998 ACM Subject Classification: F.4.3, F.1.1.
Key words and phrases: profinite topology, regular languages, uniform space, finite automata.
The author acknowledge support from the AutoMathA programme of the European Science Foundation.

The survey is organised as follows. Section 1 is a brief reminder on metric spaces. Profinite words are introduced in Section 2 and used to give equational descriptions of varieties of finite monoids in Section 3 and of lattices of regular languages in Sections 4 and 5. We discuss various extensions of the profinite metric in Section 6 and we conclude in Section 7.

1. Metric spaces

A *metric* d on a set E is a map $d : E \to \mathbb{R}_+$ from E into the set of nonnegative real numbers satisfying the three following conditions, for every $x, y, z \in E$:

(1) $d(x, y) = 0$ if and only if $x = y$,
(2) $d(y, x) = d(x, y)$,
(3) $d(x, z) \leqslant d(x, y) + d(y, z)$

An *ultrametric* satisfies the stronger property

(3′) $d(u, w) \leqslant \max\{d(u, v), d(v, w)\}$.

A *metric space* is a set E together with a metric d on E. The topology defined by d is obtained by taking as a basis the *open ε-balls* defined for $x \in E$ and $\varepsilon > 0$ by $B(x, \varepsilon) = \{y \in E \mid d(x, y) < \varepsilon\}$. In other words, an *open* set is a (possibly infinite) union of open balls. The complement of an open set is called a *closed set*. A set is *clopen* if it is both open and closed. Every metric space is *Hausdorff*, which means that any two distinct points can be separated by open sets.

A *Cauchy sequence* is a sequence $(x_n)_{n\geqslant 0}$ of elements of E such that for each $\varepsilon > 0$, there exists a integer k such that, for each $n \geqslant k$ and $m \geqslant k$, $d(x_n, x_m) < \varepsilon$.

Let (E, d) and (E', d') be two metric spaces. A function φ from E into E' is said to be *uniformly continuous* if for each $\varepsilon > 0$, there exists $\delta > 0$ such that the relation $d(x, y) < \delta$ implies $d'(\varphi(x), \varphi(y)) < \varepsilon$. If φ is uniformly continuous, the image under φ of a Cauchy sequence of E is a Cauchy sequence of E'. We say that φ is a *uniform isomorphism* if it is a uniformly continuous bijection and φ^{-1} is also uniformly continuous. Two metric spaces are *uniformly isomorphic* is there is a uniform isomorphism between them.

A metric space is *complete* if every Cauchy sequence is convergent. The *completion* of a metric space E is a complete metric space $\widehat{E}$ together with an isometric embedding of E as a dense subspace of $\widehat{E}$. One can prove that every metric space admits a completion, which is unique up to uniform isomorphism. Further, if φ is a uniformly continuous function from (E, d) in a metric space (E', d'), φ admits a uniformly continuous extension $\hat{\varphi} : \widehat{E} \to E'$ and this extension is unique.

The completion of E can be constructed as follows. Let $C(E)$ be the set of Cauchy sequences in E. Define an equivalence relation $\sim$ on $C(E)$ as follows. Two Cauchy sequences $x = (x_n)_{n\geqslant 0}$ and $y = (y_n)_{n\geqslant 0}$ are equivalent if the interleave sequence $x_0, y_0, x_1, y_1, \ldots$ is also a Cauchy sequence. The completion of E is defined to be the set $\widehat{E}$ of equivalence classes of $C(E)$. The metric d on E extends to a metric on $\widehat{E}$ defined by

$$d(x, y) = \lim_{n\to\infty} d(x_n, y_n)$$

where x and y are representative Cauchy sequences of elements in $\widehat{E}$. The definition of the equivalence insures that the above definition does not depend on the choice of x and y in their equivalence class and the fact that $\mathbb{R}$ is complete ensures that the limit exists.

2. Profinite words

In this section, A denotes a finite alphabet. The set of profinite words is defined as the completion of A^* for a certain metric. One can actually choose one of two natural metrics, which define the same uniform structure. One makes use of finite automata and the other one of finite monoids.

2.1. Separating words

A deterministic finite automaton (DFA) *separates* two words if it accepts one of them but not the other. Similarly, a finite monoid M *separates* two words u and v of A^* if there is a monoid morphism $\varphi : A^* \to M$ such that $\varphi(u) \neq \varphi(v)$.

Example 2.1.

(1) The words $ababa$ and $abaa$ can be separated by a group of order 2. Indeed, let $\pi : A^* \to \mathbb{Z}/2\mathbb{Z}$ be the morphism defined by $\pi(x) = |x| \pmod 2$. Then $\pi(ababa) = 1$ and $\pi(abaa) = 0$ and hence π separates u and v.

(2) More generally, two words u and v of unequal length can be separated by a finite cyclic group. Indeed, suppose that $|u| < |v|$ and let $n = |v|$. Let $\pi : A^* \to \mathbb{Z}/n\mathbb{Z}$ be the morphism defined by $\pi(x) = |x| \pmod n$. Then $\pi(v) = 0$ but $\pi(u) \neq 0$. A similar idea can be applied if the number of occurrences of some letter a is not the same in u and v.

(3) Let U_2 be the monoid defined on the set $\{1, a, b\}$ by the operation $aa = ba = a$, $bb = ab = b$ and $1x = x1 = x$ for all $x \in \{1, a, b\}$. Let u and v be words of $\{a, b\}^*$. Then the words ua and vb can be separated by the morphism $\pi : A^* \to U_2$ defined by $\pi(a) = a$ and $\pi(b) = b$ since $\pi(ua) = a$ and $\pi(ub) = b$.

These examples are a particular case of a general result.

Proposition 2.1. *Any pair of distinct words of A^* can be separated by a finite monoid.*

Proof. Let u and v be two distinct words of A^*. Since the language $\{u\}$ is regular, there exists a morphism φ from A^* onto a finite monoid M which recognizes it, that is, such that $\varphi^{-1}(\varphi(u)) = \{u\}$. It follows that $\varphi(v) \neq \varphi(u)$ and thus φ separates u and v. □

2.2. Profinite metrics

We now define two metrics on A^* with the following idea in mind: two words are close for d_1 [d_2] if a large DFA [monoid] is required to separate them. Let us denote by $|\mathcal{A}|$ the number of states of a DFA $\mathcal{A}$. Given two words $u, v \in A^*$, we set

$$r_1(u, v) = \min \{|\mathcal{A}| \mid \mathcal{A} \text{ is a DFA that separates } u \text{ and } v\}$$
$$r_2(u, v) = \min \{|M| \mid M \text{ is a monoid that separates } u \text{ and } v\}$$

We also set $d_1(u, v) = 2^{-r_1(u,v)}$ and $d_2(u, v) = 2^{-r_2(u,v)}$ with the usual conventions $\min \emptyset = +\infty$ and $2^{-\infty} = 0$.

Proposition 2.2. *Let d be one of the functions d_1 or d_2. Then d is an ultrametric and it satisfies the relations $d(uw, vw) \leqslant d(u, v)$ and $d(wu, wv) \leqslant d(u, v)$ for all $u, v, w \in A^*$.*

Note that the topology induced on A^* by d_1 or d_2 is discrete: every subset of A^* is clopen. Further, d_1 and d_2 define the same uniform structure.

Proposition 2.3. *The metrics d_1 and d_2 are uniformly equivalent. More precisely, the following relation holds: $2^{-\frac{1}{d_1}} \leqslant d_2 \leqslant d_1$.*

We let the reader verify that changing DFAs to NFAs in the definition of d_1 would also lead to a uniformly equivalent metric. Thus (A^*, d_1) and (A^*, d_2) are metric spaces, and their completion are uniformly isomorphic. In the sequel, we shall only use d_2 (rather than d_1) and simplify the notation to d.

The completion of (A^*, d), denoted by $\widehat{A^*}$, is the set of *profinite words* on the alphabet A. Let us state some useful properties.

Proposition 2.4.

(1) *The concatenation product is a uniformly continuous from $A^* \times A^*$ to A^*.*

(2) *Every morphism φ from A^* into a discrete finite monoid M is uniformly continuous.*

It follows from Proposition 2.4 and from the density of A^* in $\widehat{A^*}$ that the product on A^* can be extended by continuity to $\widehat{A^*}$. This extended product makes $\widehat{A^*}$ a topological monoid, called the *free profinite monoid*.

By the same argument, every morphism φ from A^* onto a finite monoid M extends uniquely to a uniformly continuous morphism from $\widehat{A^*}$ onto M. However, there are some noncontinuous morphisms form $\widehat{A^*}$ onto a finite monoid. For instance, the morphism φ from $\widehat{A^*}$ to $\{0, 1\}$, defined by $\varphi(u) = 1$ if $u \in A^*$ and $\varphi(u) = 0$ otherwise, is not continuous since $\varphi^{-1}(1) = A^*$ is not closed. Now, the restriction of φ to A^*, which is continuous, has a continuous extension to $\widehat{A^*}$. But this extension maps every profinite word to 1 and is therefore not equal to φ.

Another useful example is the following. The set 2^A of subsets of A is a monoid under union and the function $c : A^* \to 2^A$ defined by $c(a) = \{a\}$ is a morphism. Thus $c(u)$ is the set of letters occurring in u. Now c extends into a uniformly continuous morphism from $\hat{A}^*$ onto 2^A, also denoted c and called the *content* mapping.

Since A^* embeds naturally in $\widehat{A^*}$, every finite word is a profinite word. However, it is relatively difficult to give "concrete" examples of profinite words which are not words. One such example is the profinite word x^ω, associated with every finite word x. The formal definition is

$$x^\omega = \lim_{n \to \infty} x^{n!}$$

and is justified by the fact that the sequence $x^{n!}$ has a limit in $\widehat{A^*}$.

Proposition 2.5. *For each word x, the sequence $(x^{n!})_{n \geqslant 0}$ is a Cauchy sequence. It converges to an idempotent element of $\widehat{A^*}$.*

Proof. For the first part of the statement, it suffices to show that for $p, q \geqslant n$, $x^{p!}$ and $x^{q!}$ cannot be separated by a monoid of size $\leqslant n$. Let indeed $\varphi : A^* \to M$ be a monoid morphism, with $|M| \leqslant n$, and put $s = \varphi(x)$. Since M is finite, s has an idempotent power $e = s^r$, with $r \leqslant n$. By the choice of p and q, the integer r divides simultaneously $p!$ and $q!$. Consequently, $s^{p!} = s^{q!} = e$, which shows that M cannot separate $x^{p!}$ and $x^{q!}$.

For n large enough, we also have $\varphi(x^{n!})\varphi(x^{n!}) = ee = e = \varphi(x^{n!})$. It follows that the limit of the sequence $(x^{n!})_{n \geqslant 0}$ is idempotent. □

Note that x^ω is simply a notation and one should resist the temptation to interpret it as an infinite word. To get the right intuition, let us compute the image of x^ω under a

morphism onto in a finite monoid. Let M be a finite monoid, $\varphi : A^* \to M$ a morphism and let $s = \varphi(u)$. Then the sequence $s^{n!}$ is ultimately equal to s^ω, the unique idempotent of the subsemigroup of M generated by s. Consequently, we obtain the formula $\hat{\varphi}(x^\omega) = \varphi(x)^\omega$, which justifies the notation x^ω.

Another convenient way to define profinite words is to use projective systems (see [3] for more details). Suppose we are given, for each morphism φ from A^* onto a finite monoid M, an element x_φ of M. This system of elements is *projective* if for any surjective morphisms $\varphi : A^* \to M$ and $\pi : M \to N$, one has $x_{\pi\circ\varphi} = \pi(x_\varphi)$.

Proposition 2.6. *For each projective system of elements (x_φ), there is a unique profinite word x such that, for every morphism $\varphi : A^* \to M$, one has $\hat{\varphi}(x) = x_\varphi$. In particular, if two profinite words u and v satisfy $\hat{\varphi}(u) = \hat{\varphi}(v)$ for all morphisms φ onto a finite monoid, then they are equal.*

We now state the most important topological property of $\widehat{A^*}$.

Theorem 2.7. *The set of profinite words $\widehat{A^*}$ is compact.*

 If A is infinite, a profinite uniform structure can also be defined on A^* and its completion is still a compact space. However, this space is not metrizable anymore.

What about sequences? First, every profinite word is the limit of a Cauchy sequence of words. Next, a sequence of profinite words $(u_n)_{n\geqslant 0}$ is converging to a profinite word u if and only if, for every morphism φ from A^* onto a finite monoid, $\hat{\varphi}(u_n)$ is ultimately equal to $\hat{\varphi}(u)$.

Here is another example. Recall that a nonempty subset I of a monoid M is an *ideal* if, for each $s \in I$ and $x, y \in M$, $xsy \in I$. One can show that any finite monoid and any compact monoid has a unique minimal ideal (for inclusion), called *the* minimal ideal of M.

Let us fix a total order on the alphabet A and let $u_0, u_1, \ldots$ be the ordered sequence of all words of A^* in the induced shortlex order. For instance, if $A = \{a, b\}$ with $a < b$, the first elements of this sequence would be

$$1, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa, \ldots$$

It is proved in [32, 4] that the sequence of words $(v_n)_{n\geqslant 0}$ defined by

$$v_0 = u_0,\ v_{n+1} = (v_n u_{n+1} v_n)^{(n+1)!}$$

converges to a profinite word ρ_A, which is idempotent and belongs to the minimal ideal of $\widehat{A^*}$. We shall meet again this profinite word at the end of Section 5.2.

3. Equational definitions of varieties

A *variety of monoids* is a class of monoids closed under taking submonoids, quotients and direct products. Similarly, a *variety of finite monoids* is a class of finite monoids closed under taking submonoids, quotients and finite direct products. For instance, finite groups form a variety of finite monoids (the trick is that a submonoid of a finite group is a group). Another famous example is the variety of finite aperiodic monoids. Recall that a finite monoid M is *aperiodic* if there exists a positive integer n such that, for all $x \in M$, $x^n = x^{n+1}$.

Formally, an *identity* is a pair (u, v) of words of A^*, for some finite alphabet A. A monoid M satisfies the identity $u = v$ if, for every morphism $\varphi : A^* \to M$, $\varphi(u) = \varphi(v)$. It is a well known theorem of Birkhoff that varieties can be defined by a set of identities. A variety that can be defined by a finite set of identities is said to be *finitely based*. For instance, the variety of commutative monoids is finitely based, since it is defined by the single identity $xy = yx$. But in general, a variety is not finitely based, even if it is generated by a finite monoid. Consider the monoid $M = \{1, a, b, ab, ba, 0\}$ defined by the relations $aa = bb = 0$, $aba = a$ and $bab = b$. It has been proved that the variety generated by M is not finitely based.

An interesting question is to know whether varieties of finite monoids can also be defined by identities. The problem was solved by several authors but the most satisfactory answer is due to Reiterman [33]. A *profinite identity* is a pair (u, v) of profinite words of $\widehat{A^*}$, for some finite alphabet A. A finite monoid M satisfies the profinite identity $u = v$ if, for every morphism $\varphi : A^* \to M$, $\hat{\varphi}(u) = \hat{\varphi}(v)$. Reiterman's theorem is now the exact counterpart of Birkhoff's theorem:

Theorem 3.1. *Every variety of finite monoids can be defined by a set of profinite identities.*

For instance the variety of finite aperiodic monoids is defined by the identity $x^\omega = x^{\omega+1}$ and the variety of finite groups is defined by the identity $x^\omega = 1$.

4. Recognizable languages and clopen sets

A series of results, mainly due to Almeida [1, 3], [2, Theorem 3.6.1] and Pippenger [31], establishes a strong connection between regular languages and clopen sets. This section gives a short overview of these results.

Recall that a subset P of a monoid M is *recognizable* if there exists a morphism φ from M onto a finite monoid F such that $P = \varphi^{-1}(\varphi(P))$. For instance, the recognizable subsets of a free monoid are the regular languages.

The *syntactic congruence* of P is the congruence $\sim_P$ defined on M by $u \sim_P v$ if and only if, for all $x, y \in M$, the conditions $xuy \in P$ and $xvy \in P$ are equivalent. The monoid $M/\sim_P$ is called the *syntactic monoid* of P.

In the context of uniform spaces, the morphisms are uniformly continuous. It is therefore natural to extend the notion of recognizable set as follows: A subset P of a compact monoid M is *recognizable* if there exists a *uniformly continuous* morphism φ from M onto a finite discrete monoid F such that $P = \varphi^{-1}(\varphi(P))$. When M is a free profinite monoid, the recognizable subsets have a nice topological characterization, due to Hunter [14, Lemma 4].

Proposition 4.1. *Let P be a subset of $\widehat{A^*}$. The following conditions are equivalent:*

(1) *P is clopen,*

(2) *the syntactic congruence of P is a clopen subset of $\widehat{A^*} \times \widehat{A^*}$,*

(3) *P is recognizable (in the topological sense).*

Proof. Let us denote by $\sim_P$ the syntactic congruence of P and by $\hat{\eta} : \widehat{A^*} \to M$ its syntactic morphism. Recall that $s \sim_P t$ if, for all $u, v \in \widehat{A^*}$, the conditions $usv \in P$ and $utv \in P$ are equivalent.

(1) implies (2). It follows from the definition of $\sim_P$ that

$$\sim_P = \bigcap_{u,v\in\widehat{A^*}} \left((u^{-1}Pv^{-1} \times u^{-1}Pv^{-1}) \cup (u^{-1}P^cv^{-1} \times u^{-1}P^cv^{-1})\right) \tag{4.1}$$

If P is clopen, each set $u^{-1}Pv^{-1}$ is also clopen. Indeed, $u^{-1}Pv^{-1}$ is the inverse image of the clopen set P under the continuous function $x \mapsto uxy$. Now, Formula (4.1) shows that $\sim_P$ is closed.

In order to show that the complement of $\sim_P$ is closed, consider a sequence (s_n, t_n) of elements of $(\sim_P)^c$, converging to a limit (s, t). Since $s_n \not\sim_P t_n$, there exist some profinite words u_n, v_n such that $u_n s_n v_n \in P$ and $u_n t_n v_n \notin P$. Since $\widehat{A^*} \times \widehat{A^*}$ is compact, the sequence (u_n, v_n) has a convergent subsequence. Let (u, v) be its limit. Since both P and P^c are closed and since the multiplication in $\widehat{A^*}$ is continuous, one gets $usv \in P$ and $utv \notin P$. Therefore, $s \not\sim_P t$, which shows that $(\sim_P)^c$ is closed. Thus $\sim_P$ is clopen.

(2) implies (3). If $\sim_P$ is clopen, then for each $s \in \widehat{A^*}$, there exists an open neighbourhood U of s such that $U \times U \subseteq \sim_P$. Therefore U is contained in the $\sim_P$-class of s. This proves that the $\sim_P$-classes form an open partition of $\widehat{A^*}$. By compactness, this partition is finite and thus P is recognizable. Further, since each $\sim_P$-class is open, the syntactic morphism of P is continuous.

(3) implies (1). Let $\pi : \widehat{A^*} \to M$ be the syntactic morphism of P. Since P is recognizable, M is finite. One has $P = \pi^{-1}(\pi(P))$ and since M is finite, $\pi(P)$ is clopen in M. Finally, since π is continuous, P is clopen in $\widehat{A^*}$. □

We now turn to languages of A^*.

Proposition 4.2. *If L be a language of A^*, then $L = \overline{L} \cap A^*$. Further, the following conditions are equivalent:*

(1) *L is recognizable,*

(2) *$L = K \cap A^*$ for some clopen subset K of $\widehat{A^*}$,*

(3) *$\overline{L}$ is clopen in $\widehat{A^*}$,*

(4) *$\overline{L}$ is recognizable in $\widehat{A^*}$ (in the topological sense).*

Proof. The inclusion $L \subseteq \overline{L} \cap A^*$ is obvious. Let $u \in \overline{L} \cap A^*$ and let M be the syntactic monoid of $\{u\}$. Since M separates u from any word v different from u, one gets $r(u, v) \leqslant |M|$ if $u \neq v$. Let $(u_n)_{n\in\mathbb{N}}$ be a sequence of words of L converging to u. If $d(u_n, u) < 2^{-|M|}$, one has necessarily $u = u_n$ and thus $u \in L$.

(1) implies (2). If L is recognizable, there is a morphism φ from A^* onto a finite monoid M such that $L = \varphi^{-1}(\varphi(L))$. Let $K = \hat{\varphi}^{-1}(\varphi(L))$. Since M is discrete, $\varphi(L)$ is a clopen subset of M and since $\hat{\varphi}^{-1}$ is continuous, K is also clopen. Further, φ and $\hat{\varphi}$ coincide on A^* and thus $L = \hat{\varphi}^{-1}(\varphi(L)) \cap A^* = K \cap A^*$.

(2) implies (3). Suppose that $L = K \cap A^*$ with K clopen. Since K is open and A^* is dense in $\widehat{A^*}$, $K \cap A^*$ is dense in K. Thus $\overline{L} = \overline{K \cap A^*} = \overline{K} = K$. Thus $\overline{L}$ is clopen in $\widehat{A^*}$.

(3) implies (4) follows from Proposition 4.1.

(4) implies (1). Let $\widehat{\eta} : \widehat{A^*} \to F$ be the syntactic morphism of $\overline{L}$ and let $P = \widehat{\eta}(\overline{L})$. Let η be the restriction of $\widehat{\eta}$ to A^*. Then we have $L = \overline{L} \cap A^* = \widehat{\eta}^{-1}(P) \cap A^* = \eta^{-1}(P)$. Thus L is recognizable. □

We now describe the closure in $\widehat{A^*}$ of a recognizable language of A^*.

Proposition 4.3. *Let L be a regular language of A^* and let $u \in \widehat{A^*}$. The following conditions are equivalent:*

(1) $u \in \overline{L}$,

(2) $\hat{\varphi}(u) \in \varphi(L)$, *for all morphisms φ from A^* onto a finite monoid,*

(3) $\hat{\varphi}(u) \in \varphi(L)$, *for some morphism φ from A^* onto a finite monoid that recognizes L,*

(4) $\hat{\eta}(u) \in \eta(L)$, *where η is the syntactic morphism of L.*

Proof. (1) implies (2). Let φ be a morphism from A^* onto a finite monoid F and let $\hat{\varphi}$ be its continuous extension to $\widehat{A^*}$. Then $\hat{\varphi}(\overline{L}) \subset \overline{\hat{\varphi}(L)}$ since $\hat{\varphi}$ is continuous, and $\overline{\hat{\varphi}(L)} = \hat{\varphi}(L) = \varphi(L)$ since F is discrete. Thus if $u \in \overline{L}$, then $\hat{\varphi}(u) \in \varphi(L)$.

(2) implies (4) and (4) implies (3) are trivial.

(3) implies (1). Let φ be a morphism from A^* onto a finite monoid F. Let u_n be a sequence of words of A^* converging to u. Since $\hat{\varphi}$ is continuous, $\hat{\varphi}(u_n)$ converges to $\hat{\varphi}(u)$. But since F is discrete, $\hat{\varphi}(u_n)$ is actually ultimately equal to $\hat{\varphi}(u_n)$. Thus for n large enough, one has $\hat{\varphi}(u_n) = \hat{\varphi}(u)$. It follows by (3) that $\varphi(u_n) = \hat{\varphi}(u_n) \in \varphi(L)$ and since φ recognizes L, we finally get $u_n \in \varphi^{-1}(\varphi(L)) = L$. Therefore $u \in \overline{L}$. □

Let us denote by $\mathrm{Clopen}(\widehat{A^*})$ the Boolean algebra of all clopen sets of $\widehat{A^*}$.

Theorem 4.4. *The maps $L \mapsto \overline{L}$ and $K \mapsto K \cap A^*$ define mutually inverse isomorphism between the Boolean algebras* $\mathrm{Reg}(A^*)$ *and* $\mathrm{Clopen}(\widehat{A^*})$. *In particular, the following formulas hold, for all $L, L_1, L_2 \in \mathrm{Reg}(A^*)$:*

(1) $\overline{L^c} = (\overline{L})^c$,

(2) $\overline{L_1 \cup L_2} = \overline{L}_1 \cup \overline{L}_2$,

(3) $\overline{L_1 \cap L_2} = \overline{L}_1 \cap \overline{L}_2$.

Proof. Property (1) follows from Proposition 4.3. Indeed, let η be the syntactic morphism of L. Then since $L = \eta^{-1}(\eta(L))$ and $L^c = \eta^{-1}(\eta(L)^c)$, one has $\eta(L^c) = \eta(L)^c$. Therefore, one gets the following sequence of equalities:

$$\overline{L^c} = \hat{\eta}^{-1}(\eta(L^c)) = \hat{\eta}^{-1}(\eta(L)^c) = [\hat{\eta}^{-1}(\eta(L))]^c = (\overline{L})^c$$

Property (2) is a general result of topology and (3) is a consequence of (1) and (2). □

Theorem 4.4 shows that the closure operator behaves nicely with respect to Boolean operations. It also behaves nicely for the left and right quotients and for inverse of morphisms.

Proposition 4.5. *Let L be a regular language of A^* and let $x, y \in A^*$. Then $\overline{x^{-1}Ly^{-1}} = x^{-1}\overline{L}y^{-1}$.*

Proposition 4.6. *Let $\varphi : A^* \to B^*$ be a morphism of monoids and L be a regular language of B^*. Then $\hat{\varphi}^{-1}(\overline{L}) = \overline{\varphi^{-1}(L)}$.*

5. Equational characterization of languages

A *lattice of languages* of A^* is a set of regular languages of A^* containing the empty language $\emptyset$, the full language A^* and which is closed under finite union and finite intersection. The aim of this section is to show that each lattice can be, in a certain sense, defined by a

set of profinite equations. These results were obtained jointly with Mai Gehrke and Serge Grigorieff and first presented at ICALP'08 [12].

5.1. Lattices of languages

Formally, an *explicit equation* is a pair (u, v) of words of A^* and a *profinite equation* is a pair (u, v) of profinite words of $\widehat{A^*}$. We say that a language L of A^* *satisfies the explicit equation* $u \to v$ if the condition $u \in L$ implies $v \in L$ and that it *satisfies the profinite equation* $u \to v$ if the condition $u \in \overline{L}$ implies $v \in \overline{L}$. Since $\overline{L} \cap A^* = L$, the two definitions are consistent, that is, one can really consider explicit equations as a special case of profinite equations. Proposition 4.3 leads immediately to some equivalent definitions:

Corollary 5.1. *Let L be a regular language of A^*, let η be its syntactic morphism and let φ be any morphism onto a finite monoid recognizing L. The following conditions are equivalent:*

(1) *L satisfies the equation $u \to v$,*
(2) *$\hat{\eta}(u) \in \eta(L)$ implies $\hat{\eta}(v) \in \eta(L)$,*
(3) *$\hat{\varphi}(u) \in \varphi(L)$ implies $\hat{\varphi}(v) \in \varphi(L)$.*

Given a set E of equations of the form $u \to v$, the subset of $\mathrm{Reg}(A^*)$ *defined by* E is the set of all regular languages of A^* satisfying all the equations of E. It is easy to see that it is a lattice of languages.

Our aim is now to show that the converse also holds. We start with a result on languages interesting on its own right. Note in particular that there is no regularity assumption in this proposition.

Proposition 5.2. *Let L, $L_1, \ldots, L_n$ be languages. If L satisfies all the explicit equations satisfied by $L_1, \ldots, L_n$, then L belongs to the lattice of languages generated by $L_1, \ldots, L_n$.*

Proof. We claim that

$$L = \bigcup_{I \in \mathcal{I}} \bigcap_{i \in I} L_i \tag{5.1}$$

where $\mathcal{I}$ is the set of all subsets of $\{1, \ldots, n\}$ for which there exists a word $v \in L$ such that $v \in L_i$ if and only if $i \in I$. Let R be the right member of (5.1). If $u \in L$, let $I = \{i \mid u \in L_i\}$. By construction, $I \in \mathcal{I}$ and $u \in \cap_{i \in I} L_i$. Thus $u \in R$. This proves the inclusion $L \subseteq R$.

To prove the opposite direction, consider a word $u \in R$. By definition, there exists a set $I \in \mathcal{I}$ such that $u \in \cap_{i \in I} L_i$ and a word $v \in L$ such that $v \in L_i$ if and only if $i \in I$. We claim that the equation $v \to u$ is satisfied by each language L_i. Indeed, if $i \in I$, then $u \in L_i$ by definition. If $i \notin I$, then $v \notin L_i$ by definition of I, which proves the claim. It follows that $v \to u$ is also satisfied by L. Since $v \in L$, it follows that $u \in L$. This concludes the proof of (5.1) and shows that L belongs to the lattice of languages generated by $L_1, \ldots, L_n$. □

It follows that finite lattices of languages can be defined by explicit equations.

Corollary 5.3. *A finite set of languages of A^* is a lattice of languages if and only if it can be defined by a set of explicit equations of the form $u \to v$, where $u, v \in A^*$.*

Proof. Consider a finite lattice $\mathcal{L}$ of languages and let E be the set of explicit equations satisfied by all the languages of $\mathcal{L}$. Proposition 5.2 shows that any language L that satisfies the equations of E belongs to $\mathcal{L}$. Thus $\mathcal{L}$ is defined by E. □

We now are now ready for the main result.

Theorem 5.4. *A set of regular languages of A^* is a lattice of languages if and only if it can be defined by a set of equations of the form $u \to v$, where $u, v \in \widehat{A^*}$.*

Proof. For each regular language L, set

$$E_L = \{(u, v) \in \widehat{A^*} \times \widehat{A^*} \mid L \text{ satisfies } u \to v\}$$

Lemma 5.5. *For each regular language L, E_L is a clopen subset of $\widehat{A^*} \times \widehat{A^*}$.*

Proof. One has

$$\begin{aligned} E_L &= \{(u, v) \in \widehat{A^*} \times \widehat{A^*} \mid L \text{ satisfies } u \to v\} \\ &= \{(u, v) \in \widehat{A^*} \times \widehat{A^*} \mid u \in \overline{L} \text{ implies } v \in \overline{L}\} \\ &= \{(u, v) \in \widehat{A^*} \times \widehat{A^*} \mid v \in \overline{L} \text{ or } u \notin \overline{L}\} \\ &= (\overline{L}^c \times \widehat{A^*}) \cup (\widehat{A^*} \times \overline{L}) \end{aligned}$$

The result follows since, by Proposition 4.2, $\overline{L}$ is clopen. □

Let $\mathcal{L}$ be a lattice of languages and let E be the set of profinite equations satisfied by all languages of $\mathcal{L}$. We claim that E defines $\mathcal{L}$. First, by definition, every language of $\mathcal{L}$ satisfies the equations of E. It just remains to proving that if a language L satisfies the equations of E, then L belongs to $\mathcal{L}$.

First observe that the set

$$E_L \cup \{E_K^c \mid K \in \mathcal{L}\}$$

is a covering of $\widehat{A^*} \times \widehat{A^*}$. Indeed, if $(u, v) \notin \cup_{K \in \mathcal{L}} E_K^c$, then $(u, v) \in \cap_{K \in \mathcal{L}} E_K$, which means by definition that all the languages of $\mathcal{L}$ satisfy $u \to v$. It follows that L also satisfies this equation, and thus $(u, v) \in E_L$. Further, Proposition 5.5 shows that the elements of this covering are open sets. Since $\widehat{A^*} \times \widehat{A^*}$ is compact, it admits a finite subcovering, and we may assume that this covering contains E_L and is equal to

$$E_L \cup \{E_{L_1}^c, \ldots, E_{L_n}^c\}$$

for some languages $L_1, \ldots, L_n$ of $\mathcal{L}$. By the same argument as above, it follows that if an equation $u \to v$ is satisfied by $L_1, \ldots, L_n$, then it is satisfied by L. By Proposition 5.2, L belongs to the lattice of languages generated by $L_1, \ldots, L_n$ and hence belongs to $\mathcal{L}$. □

Writing $u \leftrightarrow v$ for ($u \to v$ and $v \to u$), we get an equational description of the Boolean algebras of languages.

Corollary 5.6. *A set of regular languages of A^* is a Boolean algebra of languages if and only if it can be defined by a set of profinite equations of the form $u \leftrightarrow v$, where $u, v \in \widehat{A^*}$.*

These results apply in particular to any class of regular languages defined by a fragment of logic closed under conjunctions and disjunctions (first order, monadic second order, temporal, etc.). Consider for instance Büchi's *sequential calculus*, which comprises the relation symbols S and $<$ and a predicate $\mathbf{a}$ for each letter a. To each word nonempty word $u \in A^*$ is associated a structure

$$\mathcal{M}_u = (\{1, 2, \ldots, |u|\}, S, (\mathbf{a})_{a \in A})$$

where S denotes the successor relation on $\{1, 2, \ldots, |u|\}$, $<$ is the usual order and $\mathbf{a}$ is set of all positions i such that the i-th letter of u is an a. For instance, if $A = \{a, b\}$ and $u = abaab$, then $\mathbf{a} = \{1, 3, 4\}$ and $\mathbf{b} = \{2, 5\}$. The language defined by a sentence φ is

the set of words which satisfy φ. Several fragments will be considered in this survey. We use a transparent notation of the form *Type*[Signature] to designate these fragments. For instance $FO[<]$ denotes the set of first order formulas in the signature $<$ and $\mathcal{B}\Sigma_1[S]$ consists of the Boolean combinations of existential first order formulas in the signature S.

This latter fragment allows to specify some combinatorial properties of words, like "the factor aa occurs at least twice", which defines the language $A^*aaA^*aaA^* \cup A^*aaaA^*$. Indeed, this language is described by the formula

$$\varphi = \exists x_1\ \exists x_2\ \exists y_1\ \exists y_2\ (\neg(x_1 = y_1) \wedge Sx_1x_2 \wedge Sy_1y_2 \wedge \mathbf{a}x_1 \wedge \mathbf{a}x_2 \wedge \mathbf{a}y_1 \wedge \mathbf{a}y_2$$

The $\mathcal{B}\Sigma_1(S)$-definable languages form a lattice of languages. An equational description of these languages can be derived from the results of [25]: for all $r, s, u, v, x, y \in A^*$,

$$ux^\omega y \leftrightarrow ux^{\omega+1}v \qquad ux^\omega ry^\omega sx^\omega ty^\omega v \leftrightarrow ux^\omega ty^\omega sx^\omega ry^\omega y$$
$$x^\omega uy^\omega vx^\omega \leftrightarrow y^\omega vx^\omega uy^\omega \qquad y(xy)^\omega \leftrightarrow (xy)^\omega \leftrightarrow (xy)^\omega x$$

Note that this example because it does not enter in the category considered in the next section since the correspondence lattice of languages is not closed under quotient.

We now specialize Theorems 5.4 and Corollary 5.6 to lattices of languages closed under quotient in Section 5.2 and to varieties and $\mathcal{C}$-varieties of languages in Section 5.3.

5.2. Lattices of languages closed under quotient

We say that a lattice of regular languages $\mathcal{L}$ is *closed under quotient* if for every $L \in \mathcal{L}$ and $u \in A^*$, $u^{-1}L$ and Lu^{-1} are also in $\mathcal{L}$. The equational description of such lattices can be simplified by introducing a convenient definition.

Let u and v be two profinite words of $\widehat{A^*}$. We say that L *satisfies the equation* $u \leqslant v$ if, for all $x, y \in \widehat{A^*}$, it satisfies the equation $xvy \to xuy$. Since A^* is dense in $\widehat{A^*}$, it is equivalent to state that L satisfies these equations only for all $x, y \in A^*$. But there is a much more convenient characterization using the syntactic ordered monoid of L.

Recall that the *syntactic preorder* of a language L is the relation $\leqslant_L$ over A^* defined by $u \leqslant_L v$ if and only if, for every $x, y \in M$,

$$xvy \in L \Rightarrow xuy \in L$$

It is easy to see that $\leqslant_L$ is a partial preorder on A^*, whose associated equivalence relation is the *syntactic congruence* of L. Therefore, $\leqslant_L$ induces a partial order on the syntactic monoid M of L, called the *syntactic order* of L. The ordered monoid $(M, \leqslant_L)$ is called the *syntactic ordered monoid* of L.

Proposition 5.7. *Let L be a regular language of A^*, let $(M, \leqslant_L)$ be its syntactic ordered monoid and let $\eta : A^* \to M$ be its syntactic morphism. Then L satisfies the equation $u \leqslant v$ if and only if $\hat{\eta}(u) \leqslant_L \hat{\eta}(v)$.*

Proof. Corollary 5.1 shows that L satisfies the equation $u \leqslant v$ if and only if, for every $x, y \in A^*$, $\hat{\eta}(xvy) \in \eta(L)$ implies $\hat{\eta}(xuy) \in \eta(L)$. Since $\hat{\eta}(xvy) = \hat{\eta}(x)\hat{\eta}(v)\hat{\eta}(y) = \eta(x)\hat{\eta}(v)\eta(y)$ and since η is surjective, this is equivalent to saying that, for all $s, t \in M$, $s\hat{\eta}(v)t \in \eta(L)$ implies $s\hat{\eta}(u)t \in \eta(L)$, which exactly means that $\hat{\eta}(u) \leqslant_L \hat{\eta}(v)$. □

We can now state the equational characterization of lattices of languages closed under quotients.

Theorem 5.8. *A set of regular languages of A^* is a lattice of languages closed under quotients if and only if it can be defined by a set of equations of the form $u \leqslant v$, where $u, v \in \widehat{A^*}$.*

Theorem 5.8 can be readily extended to Boolean algebras. Let u and v be two profinite words. We say that a regular language L *satisfies the equation* $u = v$ if it satisfies the equations $u \leqslant v$ and $v \leqslant u$. Proposition 5.7 now gives immediately:

Proposition 5.9. *Let L be a regular language of A^* and let η be its syntactic morphism. Then L satisfies the equation $u = v$ if and only if $\hat{\eta}(u) = \hat{\eta}(v)$.*

This leads to the following equational description of the Boolean algebras of languages closed under quotients.

Corollary 5.10. *A set of regular languages of A^* is a Boolean algebra of languages closed under quotients if and only if it can be defined by a set of equations of the form $u = v$, where $u, v \in \widehat{A^*}$.*

Let us illustrate these results by three examples taken from [12].

(1) A *language with zero* is a language whose syntactic monoid has a zero. Languages with zero form a lattice of languages closed under quotient. They are characterized by the equations $x\rho_A = \rho_A = \rho_A x$ for all $x \in A^*$, where ρ_A is the profinite word defined at the end of Section 2.

(2) A language L of A^* is *dense* if, for every word $u \in A^*$, $L \cap A^*uA^* \neq \emptyset$. One can show that regular nondense or full languages form a lattice of languages closed under quotients. They are characterized by the equations $x \leqslant \rho_A$ and $x\rho_A = \rho_A = \rho_A x$ for all $x \in A^*$.

(3) Recall that a language L is *sparse* if it has a polynomial density, that is, if $|L \cap A^n| = O(n^k)$ for some $k > 0$. Equivalently, a language is sparse if it is a finite union of languages of the form $u_0 v_1^* u_1 \cdots v_n^* u_n$, where u_0, v_1, ..., v_n, u_n are words. Sparse or full languages form a lattice of languages closed under quotient and thus admit an equational description. On a one letter alphabet, every recognizable language is sparse and the result is trivial. If $|A| \geqslant 2$, one can take the following set of equations: $x\rho_A = \rho_A = \rho_A x$ for all $x \in A^*$ and $(x^\omega y^\omega)^\omega = \rho_A$ for each $x, y \in A^+$ such that the first letter of x is different from the first letter of y.

5.3. Varieties of languages

A *class of languages* $\mathcal{F}$ associates with each alphabet A a set $\mathcal{F}(A^*)$ of regular languages of A^*. A *positive variety of languages* is a class of languages $\mathcal{V}$ such that

(1) for each alphabet A, $\mathcal{V}(A^*)$ is a lattice of languages closed under quotient,

(2) for each morphism of monoid $\varphi : A^* \to B^*$, $X \in \mathcal{V}(B^*)$ implies $\varphi^{-1}(X) \in \mathcal{V}(A^*)$,

A *variety* of languages is a positive variety of languages closed under complement.

For [positive] varieties, is is wise to use *identities* as we did for Reiterman's theorem. Intuitively, an identity is an equation in which one can substitute a word for each letter. More formally, let u and v be two profinite words of $\widehat{B}^*$ and let L be a regular language of A^*. One says that L *satisfies the profinite identity* $u \leqslant v$ $[u = v]$ if, for all morphisms $\gamma : B^* \to A^*$, L satisfies the equation $\hat{\gamma}(u) \leqslant \hat{\gamma}(v)$ $[\hat{\gamma}(u) = \hat{\gamma}(v)]$.

Theorem 5.11. *A class of languages is a positive variety of languages if and only if it can be defined by a set of profinite identities of the form $u \leqslant v$. It is a variety of languages if and only if it can be defined by a set of profinite identities of the form $u = v$.*

Theorem 5.11 and Reiterman's theorem allows one to recover Eilenberg's variety theorem. Let us first recall this important result.

If $\mathbf{V}$ is a variety of finite monoids, denote by $\mathcal{V}(A^*)$ the set of regular languages of A^* whose syntactic monoid belongs to $\mathbf{V}$. The correspondence $\mathbf{V} \to \mathcal{V}$ associates with each variety of finite monoids a variety of languages. Conversely, to each variety of languages $\mathcal{V}$, we associate the variety of monoids $\mathbf{V}$ generated by the monoids of the form $M(L)$ where $L \in \mathcal{V}(A^*)$ for a certain alphabet A. Eilenberg's variety theorem [10] states that the correspondences $\mathbf{V} \to \mathcal{V}$ and $\mathcal{V} \to \mathbf{V}$ define mutually inverse bijective correspondences between varieties of finite monoids and varieties of languages.

Now, it follows from Theorem 5.11 that any variety of languages can be defined by a set of profinite identities. And Reiterman's theorem states that varieties of finite monoids can also be defined by profinite identities. This gives the variety theorem.

There is an analogous result for ordered monoids [22], which gives mutually inverse bijective correspondences between the varieties of finite ordered monoids and the positive varieties of languages.

Equational descriptions are known for a large number of [positive] varieties of languages. We just give a few emblematic examples in elliptic style below, but many more can be found in the survey articles [9, 24].

(1) Finite or full languages: $yx^\omega = x^\omega = x^\omega y$ and $y \leqslant x^\omega$.
(2) Star-free languages (closure of finite languages under Boolean operations and product): $x^{\omega+1} = x^\omega$. These languages are also captured by the logical fragment $FO[<]$.
(3) Shuffle ideals (finite unions of languages of the form $A^*a_1A^*a_2A^* \cdots a_kA^*$, where $a_1, \ldots, a_k$ are letters): $x \leqslant 1$. These languages are also captured by the logical fragment $\Sigma_1[<]$.
(4) Piecewise testable languages (the Boolean closure of shuffle ideals): $x^{\omega+1} = x^\omega$ and $(xy)^\omega = (yx)^\omega$. These languages are also captured by the logical fragment $\mathcal{B}\Sigma_1[<]$.
(5) Unambiguous star-free languages (closure of finite languages under Boolean operations and unambiguous product): $x^{\omega+1} = x^\omega$ and $(xy)^\omega(yx)^\omega(xy)^\omega = (xy)^\omega$. These languages are also captured by the logical fragments $FO_2[<]$ (first order with two variables), by $\Delta_2[<]$ or by unary temporal logic (based on the operators *eventually in the future* and *eventually in the past*). Finally, they are disjoint unions of unambiguous products of the form $A_0^*a_1A_1^* \cdots a_kA_k^*$, where $a_1, \ldots, a_k$ are letters and $A_0, \ldots, A_k$ are subsets of A.

A more general notion was introduced by Straubing [39] (see also Ésik and Ito [11]). Let $\mathcal{C}$ be a class of morphisms between finitely generated free monoids that is closed under composition and contains all length-preserving morphisms. Examples include the classes of *length-preserving* morphisms, of *length-multiplying* morphisms, of *non-erasing* morphisms, of *length-decreasing* morphisms and of course the class of all morphisms.
A *positive $\mathcal{C}$-variety of languages* is a class of languages $\mathcal{V}$ such that

(1) for every alphabet A, $\mathcal{V}(A^*)$ is a lattice of languages closed under quotient,
(2) if $\varphi\colon A^* \to B^*$ is a morphism of $\mathcal{C}$, $L \in \mathcal{V}(B^*)$ implies $\varphi^{-1}(L) \in \mathcal{V}(A^*)$,

A *$\mathcal{C}$-variety of languages* is a positive $\mathcal{C}$-variety of languages closed under complement.

It is easy to extend Theorem 5.11 to $\mathcal{C}$-varieties by using $\mathcal{C}$-identities. Let us say that a language L *satisfies the profinite $\mathcal{C}$-identity* $u \leqslant v$ $[u = v]$ if, for all $\mathcal{C}$-morphisms $\gamma : B^* \to A^*$, L satisfies the equation $\hat{\gamma}(u) \leqslant \hat{\gamma}(v)$ $[\hat{\gamma}(u) = \hat{\gamma}(v)]$. Then we can state

Theorem 5.12. *A class of languages of A^* is a positive $\mathcal{C}$-variety of languages if and only if it can be defined by a set of profinite $\mathcal{C}$-identities of the form $u \leqslant v$. It is a $\mathcal{C}$-variety of languages if and only if it can be defined by a set of profinite $\mathcal{C}$-identities of the form $u = v$.*

5.4. Summary

We summarize below the various types of equations...

Closed under	Equations	Definition
$\cup, \cap$	$u \to v$	$\hat{\eta}(u) \in \hat{\eta}(L) \Rightarrow \hat{\eta}(v) \in \hat{\eta}(L)$
quotient	$u \leqslant v$	$xvy \to xuy$
complement	$u \leftrightarrow v$	$u \to v$ and $v \to u$
quotient and complement	$u = v$	$xvy \leftrightarrow xuy$

...and the various types of $\mathcal{C}$-identities.

Class of morphisms $\mathcal{C}$	Interpretation of variables
all morphisms	words
nonerasing morphisms	nonempty words
length multiplying morphisms	words of equal length
length preserving morphisms	letters

6. Pro-V uniformities

The profinite uniformities defined in Section 2 can be generalized in various ways using varieties of finite ordered monoids [30] or even lattices of languages [12]. In this section, we only consider the case of a variety of finite monoids $\mathbf{V}$.

The idea is to generalize the metric d on A^* by defining $d_{\mathbf{V}}$ as follows:

$$r_{\mathbf{V}}(u, v) = \min \{|M| \mid M \text{ is a monoid of } \mathbf{V} \text{ that separates } u \text{ and } v\}$$
$$d_{\mathbf{V}}(u, v) = 2^{-r_{\mathbf{V}}(u,v)}$$

Unfortunately, $d_{\mathbf{V}}$ is not always a metric since the monoids of $\mathbf{V}$ may not suffice to separate two distinct words. For instance, if $\mathbf{V}$ is the variety **Com** of finite commutative monoids, there is no way to separate the words ab and ba.

There are two possibilities to overcome this difficulty. The first solution is algebraic: the relation $\sim_{\mathbf{V}}$ defined on A^* by $u \sim_{\mathbf{V}} v$ if and only if $d_{\mathbf{V}}(u, v) = 0$ is a congruence on A^* and $d_{\mathbf{V}}$ induces a metric on the quotient monoid $A^*/\sim_{\mathbf{V}}$ and one can take the completion of the metric space $(A^*/\sim_{\mathbf{V}}, d_{\mathbf{V}})$. For instance, if $\mathbf{V} = \mathbf{Com}$, $A^*/\sim_{\mathbf{V}}$ is the free commutative monoid $\mathbb{N}^A$.

The second solution is topological. Even if $d_{\mathbf{V}}$ fails to be a metric, it still satisfies conditions (2) and (3) of the definition of a metric and this suffices to define an Hausdorff completion. A systematic study of the corresponding uniform spaces can be found in [30].

The two methods lead to the same object, called the free pro-$\mathbf{V}$ monoid on A, denoted by $\widehat{F}_{\mathbf{V}}(A)$. This monoid is compact and can be alternatively defined as the quotient of $\widehat{A^*}$ by the congruence induced by the profinite identities defining $\mathbf{V}$.

The study of these free profinite monoids, for various varieties $\mathbf{V}$, is a central topic of the theory of finite monoids and regular languages. It is not possible to describe in detail the numerous results obtained in this area, and we refer the interested reader to the survey articles [3, 5] and to the articles of Almeida, Auinger, Margolis, Steinberg, Weil, Zeitoun or the author for more information.

We limit ourselves in this survey to direct consequences in automata theory. First, uniform continuous functions have a very concrete characterization.

Theorem 6.1. *A function $f : A^* \to B^*$ is uniformly continuous for $d_{\mathbf{V}}$ if and only if, for every language L of B^* recognized by a monoid of $\mathbf{V}$, the language $f^{-1}(L)$ is also recognized by a monoid of $\mathbf{V}$.*

It is worth considering separately the case where $\mathbf{V}$ is the variety of all finite monoids.

Corollary 6.2. *A function $f : A^* \to B^*$ is uniformly continuous for d if and only if, for every regular language L of B^*, the language $f^{-1}(L)$ is also regular.*

We illustrate the power of this approach by solving a standard exercise in automata theory: Show that the *square root* of a regular language is regular.

Proof. Since the concatenation product is uniformly continuous, the product h of two uniformly continuous functions f and g, defined by $h(u) = f(u)g(u)$, is also uniformly continuous. In particular, the function $u \to u^2$, from A^* into itself, is uniformly continuous. It follows that, if L is regular, the set $\{u \in A^* \mid u^2 \in L\}$ is also regular. □

Here are two more advanced results. Given a class $\mathcal{L}$ of regular languages, the *polynomial closure* $\mathrm{Pol}(\mathcal{L})$ of $\mathcal{L}$ is the set of all languages which are finite unions of languages of the form $L_0a_1L_1 \cdots a_kL_k$ where $a_1, \ldots, a_k$ are letters and $L_0, \ldots, L_k$ are languages of $\mathcal{L}$. It can be shown that if $\mathcal{V}$ is a variety of languages, then $\mathrm{Pol}(\mathcal{V})$ is a positive variety of languages. Further, it follows from the results of [29] that, given the profinite equations satisfied by $\mathcal{V}$, one can find, at least implicitly, the profinite equations defining $\mathrm{Pol}(\mathcal{V})$.

Theorem 6.3. *The positive variety Pol($\mathcal{V}$) is defined by the profinite identities of the form $x^\omega y x^\omega \leqslant x^\omega$ where x and y are profinite words such that the identities $x = y = x^2$ hold in $\mathcal{V}$.*

Similar results hold for the unambiguous polynomial closure (the identities are now of the form $x^\omega y x^\omega = x^\omega$) and for the closure under Boolean operations and product.

Theorem 6.4. *The closure under Boolean operations and product of a variety of languages $\mathcal{V}$ is defined by the profinite identities of the form $x^{\omega+1} = x^\omega$ where x and y are profinite words such that the identities $x = y = x^2$ hold in $\mathcal{V}$.*

We now concentrate on two important particular cases which proved to have unexpected connections with automata theory: the variety of finite groups $\mathbf{G}$ and the variety of finite p-groups $\mathbf{G}_p$, where p is a prime number. Recall that a *p-group* is a finite group whose order is a power of p.

6.1. Progroup topology

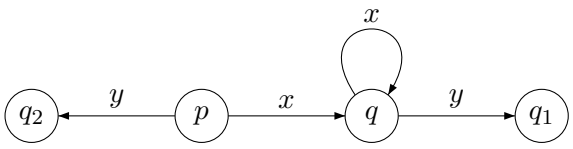
The pro-group topology was originally introduced by M. Hall in group theory [13]. It was first considered for the free monoid by Reutenauer [34, 35] and studied in full details in [19, 26, 23], notably in connection with a celebrated problem of Rhodes in finite semigroup theory. This problem was ultimately solved by Ash using a combinatorial approach [6] and by Ribes and Zalesskii using profinite methods [36].

The definition of the *pro-group metric* is obtained by taking $\mathbf{V} = \mathbf{G}$ in the definition of $d_{\mathbf{V}}$. One can show that $d_{\mathbf{G}}$ is an ultrametric, but contrary to the profinite metric d, the topology induced by $d_{\mathbf{G}}$ on A^* is not discrete and it is an interesting question to decide whether a given regular language is open, closed or clopen for this topology.

Recall that a *group language* is a language whose syntactic monoid is a group, or, equivalently, is recognized by a finite deterministic automaton in which each letter defines a permutation of the set of states. According to the definition of a polynomial closure, a *polynomial of group languages* is a finite union of languages of the form $L_0a_1L_1 \cdots a_kL_k$ where $a_1, \ldots, a_k$ are letters and $L_0, \ldots, L_k$ are group languages. It can be shown that a regular language is clopen if and only if it is a group language. For the open sets, the following characterization holds.

Theorem 6.5. *Let L be a regular language. The following conditions are equivalent:*

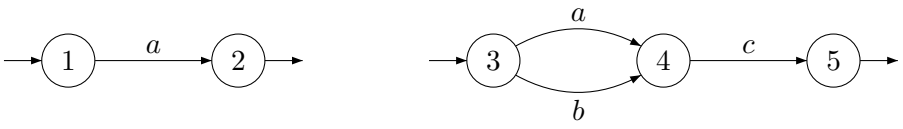
(1) *L is a polynomial of group languages,*
(2) *L is open in the group topology,*
(3) *L satisfies the identity $x^\omega \leqslant 1$,*
(4) *the minimal deterministic automaton of L contains no configuration of the form*



where $x, y \in A^*$, q_1 is final and q_2 is nonfinal.

One can show also that the closure of regular language for $d_{\mathbf{G}}$ is again regular and can be effectively computed [26, 36].

These results also permit to study another class of finite automata. A *reversible automaton* is a finite automaton whose transitions are both deterministic and co-deterministic. In other words, each letter a induces a partial one-to-one map from the set of states into itself. However, we make no assumption on the set of initial states and the set of final states, which can be arbitrary. It is not difficult to see that any finite language can be accepted by a reversible automaton. For instance, a reversible automaton accepting $\{a, ac, bc\}$ is represented below:



It is tempting to guess that a language is accepted by some reversible automaton if and only if its minimal DFA is reversible, but this is not the case and the characterization of these languages [20] is more involved.

Theorem 6.6. *Let L be a regular language and let M be its syntactic monoid. The following conditions are equivalent:*

(1) *L is accepted by a reversible automaton,*
(2) *the idempotents of M commute and L is closed in the profinite group topology of A^*.*
(3) *L satisfies the identities $x^\omega y^\omega = y^\omega x^\omega$ and $1 \leqslant ^\omega$,*

6.2. Pro-p topology

The definition of the *pro-p metric* is obtained by taking $\mathbf{V} = \mathbf{G}_p$ in the definition of $d_{\mathbf{V}}$. The resulting ultrametric d_p defines the *p-adic topology* on A^*. When A is a one letter alphabet, the free monoid A^* is isomorphic to the additive monoid $\mathbb{N}$ and its pro-p completion is the additive group of p-adic numbers.

As for $d_{\mathbf{G}}$, the closure of regular language for d_p is again regular and can be effectively computed [37, 18], but this is a difficult result.

There is also a nice connection [21] 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 [10, 15], 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. Further, if $u = a^n$ and $v = a^m$, then $\binom{v}{u} = \binom{m}{n}$ and hence these numbers constitute a generalization of the classical binomial coefficients. 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 [21, Theorem 4.4] that d'_p is an ultrametric uniformly equivalent to d_p. The next proposition should be compared with Proposition 2.5.

Proposition 6.7. *For every word $u \in A^*$, one has $\lim_{n\to\infty} u^{p^n} = 1$ for the metric d_p.*

Proof. By the definition of the topology, it suffices to show that if $\varphi : A^* \to G$ is a monoid morphism onto a discrete p-group G, then $\lim_{n\to\infty} \varphi(g^{p^n}) = 1$. But if $|G| = p^k$, then for $n \geqslant k$, $\varphi(g^{p^n}) = 1$ since the order of $\varphi(g)$ divides p^k. □

There is another nice example of converging sequence, related to the Prouhet-Thue-Morse word $t = abbabaabbaababba\cdots$. Recall that this infinite word on the alphabet $\{a, b\}$ is obtained from a by iterating the morphism τ defined by $\tau(a) = ab$ and $\tau(b) = ba$.

Denoting by $t[m]$ the prefix of length m of t, one has:

Theorem 6.8 (See [7])**.** *For every prime number p, there exists a strictly increasing sequence $m_1 < m_2 < \cdots$ such that $\lim_{n\to\infty} t[m_n] = 1$.*

The sequence m_n depends on p but can be explicitly given. For $p \neq 2$, one can choose $m_n = 2^n p^{1+\lfloor log_p n \rfloor}$, but as often in mathematics, the case $p = 2$ is singular. In this case, one can take $m_n = 2^k$ if $F_{k-1} \leqslant n < F_k$, where F denotes the Fibonacci sequence defined by $F_0 = 0$, $F_1 = 1$ and $F_{n+2} = F_{n+1} + F_n$ for every $n \geqslant 0$.

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 [10, Theorem 10.1, p. 239]). Let us call a p-group language a language recognized by a p-group.

Proposition 6.9. *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 $x \in A^*$.

We conclude this section with a result presented at STACS last year [28], which extends a classical result of Mahler [16, 17].

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

$$\Delta^w f(u) = \sum_{0 \leqslant |x| \leqslant |w|} (-1)^{|w|+|x|} \binom{w}{x} f(ux)$$

For instance, $\Delta^{aab} f(u) = -f(u) + 2f(ua) + f(ub) - f(uaa) - 2f(uab) + f(uaab)$.

One can show that for each function $f : A^* \to \mathbb{Z}$, there exists a unique family $\langle f, v\rangle_{v \in A^*}$ of integers such that, for all $u \in A^*$, $f(u) = \sum_{v \in A^*} \langle f, v\rangle \binom{u}{v}$. These coefficients are given by

$$\langle f, v\rangle = (\Delta^v f)(1) = \sum_{0 \leqslant |x| \leqslant |v|} (-1)^{|v|+|x|} \binom{v}{x} f(x)$$

If n is a non-zero integer, we denote by $|n|_p$ the *p-adic norm* of n, which is the real number p^{-k}, where k is the largest integer such that p^k divides n. By convention, $|0|_p = 0$. The main result of [28] gives a simple description of the uniformly continuous functions for d_p.

Theorem 6.10. *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 $\mathbb{Z}$. The following conditions are equivalent:*

(1) *f is uniformly continuous for d_p,*
(2) *the partial sums $\sum_{0 \leqslant |v| \leqslant n} \langle f, v\rangle \binom{u}{v}$ converge uniformly to f,*
(3) *$\lim_{|v| \to \infty} |\langle f, v\rangle|_p = 0$.*

7. Conclusion

Profinite topologies are a powerful tool to solve decidability problems on regular languages. In particular, they lead to equational definitions of lattices of languages which can sometimes be used to obtain decidability results. For instance, it follows from the deep results of McNaughton and Schützenberger that $FO[<]$-definable languages are defined by the identity $x^\omega = x^{\omega+1}$. Since Büchi has shown that monadic second order $MSO[<]$ captures all regular languages, it follows that one can effectively decide whether a monadic second order formula is equivalent to a first order formula on finite words (or, in the language of model theory, on finite coloured linear orders).

There are however two problems to extend this type of arguments to a given lattice of languages. First, one needs to find effectively the equations foretold by Theorem 5.4. This step can be extremely difficult. For instance, it is conjectured that the languages captured by the logical fragment $\mathcal{B}\Sigma_2[<]$ are defined by the identities

$$\left((x^\omega py^\omega qx^\omega)^\omega x^\omega py^\omega sx^\omega (x^\omega ry^\omega sx^\omega)^\omega\right) = \left((x^\omega py^\omega qx^\omega)^\omega (x^\omega ry^\omega sx^\omega)^\omega\right)$$

where $x, y, p, q, r, s \in \widehat{A^*}$ are profinite words with the same content, but this conjecture is still open.

If some set of equations have been found, one still needs to decide whether a given regular language satisfies these equations. This second problem might also be difficult to solve. For instance, it is not clear whether the implicit descriptions given in Theorems 6.3 ant 6.4 lead to effective decision criteria. A lot of work has been done, notably by Almeida and Steinberg, to address this type of questions. A key idea is that certain varieties can be defined by identities involving words and simple profinite operations, like the ω operation. When a basis of such identities can be found, the second problem becomes generally easy.

To conclude, we would like to suggest a new path of research. The starting point is the following observation: the hierarchies considered in computability, in complexity theory and in descriptive set theory are defined in terms of appropriate reductions. In each case, the definition of a reduction follows the same pattern: given two sets X and Y, Y reduces to X if there exists a function f such that $X = f^{-1}(Y)$. In complexity theory, f is required to be computable in polynomial time. In descriptive set theory, f is continuous. We propose to study the reductions between regular languages based on uniformly continuous functions (for instance for some metric $d_{\mathbf{V}}$). One could then explore the corresponding hierarchies, as it has been done in descriptive set theory and in computability theory.

References

[1] J. Almeida, Residually finite congruences and quasi-regular subsets in uniform algebras, *Partugaliæ Mathematica* **46** (1989), 313–328.

[2] J. Almeida, *Finite semigroups and universal algebra*, World Scientific Publishing Co. Inc., River Edge, NJ, 1994. Translated from the 1992 Portuguese original and revised by the author.

[3] J. Almeida, Profinite semigroups and applications, in *Structural theory of automata, semigroups, and universal algebra*, pp. 1–45, *NATO Sci. Ser. II Math. Phys. Chem.* vol. 207, Springer, Dordrecht, 2005. Notes taken by Alfredo Costa.

[4] J. Almeida and M. V. Volkov, Profinite identities for finite semigroups whose subgroups belong to a given pseudovariety, *J. Algebra Appl.* **2**,2 (2003), 137–163.

[5] J. Almeida and P. Weil, Relatively free profinite monoids: an introduction and examples, in *NATO Advanced Study Institute Semigroups, Formal Languages and Groups*, J. Fountain (éd.), vol. 466, pp. 73–117, Kluwer Academic Publishers, 1995.

[6] C. J. Ash, Inevitable graphs: a proof of the type II conjecture and some related decision procedures, *Internat. J. Algebra Comput.* **1**,1 (1991), 127–146.

[7] J. Berstel, M. Crochemore and J.-E. Pin, Thue sequence and p-adic topology of the free monoid, *Discrete Mathematics* **76** (1989), 89–94.

[8] G. Birkhoff, Moore-Smith convergence in general topology, *Ann. of Math. (2)* **38**,1 (1937), 39–56.

[9] M. J. J. Branco, Varieties of languages, in *Semigroups, algorithms, automata and languages (Coimbra, 2001)*, pp. 91–132, World Sci. Publ., River Edge, NJ, 2002.

[10] S. Eilenberg, *Automata, languages, and machines. Vol. B*, Academic Press [Harcourt Brace Jovanovich Publishers], New York, 1976.

[11] Z. Ésik and M. Ito, Temporal logic with cyclic counting and the degree of aperiodicity of finite automata, *Acta Cybernetica* **16** (2003), 1–28.

[12] M. Gehrke, S. Grigorieff and J.-E. Pin, Duality and equational theory of regular languages, in *ICALP 2008, Part II*, L. Aceto and al. (éd.), Berlin, 2008, pp. 246–257, *Lect. Notes Comp. Sci.* vol. 5126, Springer.
[13] M. Hall, Jr., A topology for free groups and related groups, *Ann. of Math. (2)* **52** (1950), 127–139.
[14] R. Hunter, Certain finitely generated compact zero-dimensional semigroups, *J. Austral. Math. Soc. (Series A)* **44** (1988), 265–270.
[15] M. Lothaire, *Combinatorics on words*, *Cambridge Mathematical Library*, Cambridge University Press, Cambridge, 1997. With a foreword by Roger Lyndon and a preface by Dominique Perrin, Corrected reprint of the 1983 original, with a new preface by Perrin.
[16] 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.
[17] K. Mahler, A correction to the paper "An interpolation series for continuous functions of a p-adic variable.", *J. Reine Angew. Math.* **208** (1961), 70–72.
[18] S. Margolis, M. Sapir and P. Weil, Closed subgroups in pro-$\mathbf{V}$ topologies and the extension problem for inverse automata, *Internat. J. Algebra Comput.* **11**,4 (2001), 405–445.
[19] J.-E. Pin, Topologies for the free monoid, *J. of Algebra* **137** (1991), 297–337.
[20] J.-E. Pin, On reversible automata, in *Proceedings of the first LATIN conference*, Saõ-Paulo, 1992, pp. 401–416, *Lect. Notes Comp. Sci.* n° 583, Springer.
[21] J.-E. Pin, Topologie p-adique sur les mots, *Journal de théorie des nombres de Bordeaux* **5** (1993), 263–281.
[22] J.-E. Pin, A variety theorem without complementation, *Russian Mathematics (Iz. VUZ)* **39** (1995), 80–90.
[23] J.-E. Pin, Polynomial closure of group languages and open sets of the Hall topology, *Theoret. Comput. Sci.* **169** (1996), 185–200.
[24] J.-E. Pin, Syntactic semigroups, in *Handbook of formal languages*, G. Rozenberg and A. Salomaa (éd.), vol. 1, ch. 10, pp. 679–746, Springer, 1997.
[25] J.-E. Pin, The expressive power of existential first order sentences of Büchi's sequential calculus, *Discrete Mathematics* **291** (2005), 155–174.
[26] J.-E. Pin and C. Reutenauer, A conjecture on the Hall topology for the free group, *Bull. London Math. Soc.* **23** (1991), 356–362.
[27] J.-E. Pin and P. V. Silva, A topological approach to transductions, *Theoret. Comput. Sci.* **340** (2005), 443–456.
[28] J.-E. Pin and P. V. Silva, A Mahler's theorem for functions from words to integers, in *25th International Symposium on Theoretical Aspects of Computer Science (STACS 2008)*, S. Albers and P. Weil (éd.), Dagstuhl, Germany, 2008, pp. 585–596, Internationales Begegnungs- Und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
[29] J.-E. Pin and P. Weil, Polynomial closure and unambiguous product, *Theory Comput. Systems* **30** (1997), 1–39.
[30] J.-E. Pin and P. Weil, Uniformities on free semigroups, *International Journal of Algebra and Computation* **9** (1999), 431–453.
[31] N. Pippenger, Regular languages and Stone duality, *Theory Comput. Syst.* **30**,2 (1997), 121–134.
[32] N. R. Reilly and S. Zhang, Decomposition of the lattice of pseudovarieties of finite semigroups induced by bands, *Algebra Universalis* **44**,3-4 (2000), 217–239.
[33] J. Reiterman, The Birkhoff theorem for finite algebras, *Algebra Universalis* **14**,1 (1982), 1–10.
[34] C. Reutenauer, Une topologie du monoïde libre, *Semigroup Forum* **18**,1 (1979), 33–49.
[35] C. Reutenauer, Sur mon article: "Une topologie du monoïde libre" [Semigroup Forum **18** (1979), no. 1, 33–49; MR 80j:20075], *Semigroup Forum* **22**,1 (1981), 93–95.
[36] L. Ribes and P. A. Zalesskii, On the profinite topology on a free group, *Bull. London Math. Soc.* **25**,1 (1993), 37–43.
[37] L. Ribes and P. A. Zalesskii, The pro-p topology of a free group and algorithmic problems in semigroups, *Internat. J. Algebra Comput.* **4**,3 (1994), 359–374.
[38] M. H. Stone, The representation of Boolean algebras, *Bull. Amer. Math. Soc.* **44**,12 (1938), 807–816.
[39] H. Straubing, On logical descriptions of regular languages, in *LATIN 2002*, Berlin, 2002, pp. 528–538, *Lect. Notes Comp. Sci.* n° 2286, Springer.
[40] P. Weil, Profinite methods in semigroup theory, *Int. J. Alg. Comput.* **12** (2002), 137–178.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 51–62
www.stacs-conf.org

LOWER BOUNDS FOR MULTI-PASS PROCESSING OF MULTIPLE DATA STREAMS

NICOLE SCHWEIKARDT

Institut für Informatik, Goethe-Universität Frankfurt am Main,
Robert-Mayer-Str. 11–15, D-60325 Frankfurt am Main, Germany
E-mail address: schweika@informatik.uni-frankfurt.de
URL: http://www.informatik.uni-frankfurt.de/~schweika

ABSTRACT. This paper gives a brief overview of computation models for data stream processing, and it introduces a new model for multi-pass processing of multiple streams, the so-called *mp2s-automata*. Two algorithms for solving the set disjointness problem with these automata are presented. The main technical contribution of this paper is the proof of a lower bound on the size of memory and the number of heads that are required for solving the set disjointness problem with mp2s-automata.

1. Introduction

In the basic data stream model, the input consists of a stream of data items which can be read only sequentially, one after the other. For processing these data items, a memory buffer of limited size is available. When designing data stream algorithms, one aims at algorithms whose memory size is far smaller than the size of the input.

Typical application areas for which data stream processing is relevant are, e.g., IP network traffic analysis, mining text message streams, or processing meteorological data generated by sensor networks. Data stream algorithms are also used to support query optimization in relational database systems. In fact, virtually all query optimization methods in relational database systems rely on information about the number of distinct values of an attribute or the self-join size of a relation — and these pieces of information have to be maintained while the database is updated. Data stream algorithms for accomplishing this task have been introduced in the seminal paper [2].

Most parts of the data stream literature deal with the task of performing **one pass over a single stream**. For a detailed overview on algorithmic techniques for this scenario we refer to [23]. *Lower bounds* on the size of memory needed for solving a problem by a one-pass algorithm are usually obtained by applying methods from *communication complexity* (see, e.g., [2, 20]). In fact, for many concrete problems it is known that the memory needed for solving the problem by a deterministic one-pass algorithm is at least linear in the size n of the input. For some of these problems, however, *randomized* one-pass algorithms can still compute good *approximate* answers while using memory

1998 ACM Subject Classification: F.1.1 (Computation by Abstract Devices: Models of Computation);
F.2.2 (Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems);
F.2.3 (Analysis of Algorithms and Problem Complexity: Tradeoffs between Complexity Measures) .

Key words and phrases: data streams, lower bounds, machine models, automata, the set disjointness problem.

of size sublinear in n. Typically, such algorithms are based on *sampling*, i.e., only a "representative" portion of the data is taken into account, and *random projections*, i.e., only a rough "sketch" of the data is stored in memory. See [23, 10] for a comprehensive survey of according algorithmic techniques and for pointers to the literature.

Also the generalization where **multiple passes over a single stream** are performed, has received considerable attention in the literature. Techniques for proving lower bounds in this scenario can be found, e.g., in [20, 18, 9, 12, 22].

A few articles also deal with the task of **processing several streams in parallel**. For example, the authors of [28] consider algorithms which perform one pass over several streams. They introduce a new model of multi-party communication complexity that is suitable for proving lower bounds on the amount of memory necessary for one-pass algorithms on multiple streams. In [28], these results are used for determining the exact space complexity of processing particular XML twig queries.
In recent years, the database community has also addressed the issue of designing general-purpose *data stream management systems* and query languages that are suitable for new application areas where multiple data streams have to be processed in parallel. To get an overview of this research area, [3] is a good starting point. Foundations for a theory of *stream queries* have been laid in [19]. Stream-based approaches have also been examined in detail in connection with *XML query processing and validation*, see, e.g. the papers [27, 26, 13, 8, 4, 5, 16].

The *finite cursor machines* (FCMs, for short) of [14] are a computation model for performing **multiple passes over multiple streams**. FCMs were introduced as an abstract model of database query processing. Formally, they are defined in the framework of *abstract state machines*. Informally, they can be described as follows: The input for an FCM is a relational database, each relation of which is represented by a *table*, i.e., an ordered list of rows, where each row corresponds to a tuple in the relation. Data elements are viewed as "indivisible" objects that can be manipulated by a number of "built-in" operations. This feature is very convenient to model standard operations on data types like integers, floating point numbers, or strings, which may all be part of the universe of data elements. FCMs can operate in a finite number of *modes* using an *internal memory* in which they can store bitstrings. They access each relation through a finite number of *cursors*, each of which can read one row of a table at any time. The model incorporates certain *streaming* or *sequential processing* aspects by imposing a restriction on the movement of the cursors: They can move on the tables only sequentially in one direction. Thus, once the last cursor has left a row of a table, this row can never be accessed again during the computation. Note, however, that several cursors can be moved asynchronously over the same table at the same time, and thus, entries in different, possibly far apart, regions of the table can be read and processed simultaneously.

A common feature of the computation models mentioned so far in this paper is that the input streams are *read-only* streams that cannot be modified during a pass. Recently, also **stream-based models for external memory processing** have been proposed, among them the *StrSort model* [1, 24], the *W-Stream* model [11], and the model of *read/write streams* [17, 16, 15, 7, 6]. In these models, several passes may be performed over a single stream or over several streams in parallel, and during a pass, the content of the stream may be modified.

A detailed introduction to *algorithms on data streams*, respectively, to the related area of *sublinear algorithms* can be found in [23, 10]. A survey of *stream-based models for external memory processing* and of methods for proving *lower bounds* in these models is given in [25]. A database systems oriented overview of so-called *data stream systems* can be found in [3]. For a list of *open problems* in the area of data streams we refer to [21].

In the remainder of this article, a new computation model for multi-pass processing of multiple streams is introduced: the *mp2s-automata*. In this model, (read-only) streams can be processed by forward scans as well as backward scans, and several "heads" can be used to perform several passes over the streams in parallel. After fixing the basic notation in Section 2, the computation model of mp2s-automata is introduced in Section 3. In Section 4, we consider the *set disjointness problem* and prove upper bounds as well as lower bounds on the size of memory and the number of heads that are necessary for solving this problem with an mp2s-automaton. Section 5 concludes the paper by pointing out some directions for future research.

2. Basic notation

If f is a function from the set of non-negative integers to the set of reals, we shortly write $f(n)$ instead of $\lceil f(n) \rceil$ (where $\lceil x \rceil$ denotes the smallest integer $\geqslant x$). We write $\lg n$ to denote the logarithm of n with respect to base 2. For a set $\mathbb{D}$ we write $\mathbb{D}^*$ to denote the set of all finite strings over alphabet $\mathbb{D}$. We view $\mathbb{D}^*$ as the set of all finite *data streams* that can be built from elements in $\mathbb{D}$. For a stream $\vec{S} \in \mathbb{D}^*$ write $|\vec{S}|$ to denote the length of $\vec{S}$, and we write s_i to denote the element in $\mathbb{D}$ that occurs at the i-th position in $\vec{S}$, i.e., $\vec{S} = s_1 s_2 \cdots s_{|\vec{S}|}$.

3. A computation model for multi-pass processing of multiple streams

In this section, we fix a computation model for multi-pass processing of multiple streams. The model is quite powerful: Streams can be processed by forward scans as well as backward scans, and several "heads" can be used to perform several passes over the stream in parallel. For simplicity, we restrict attention to the case where just *two* streams are processed in parallel. Note, however, that it is straightforward to generalize the model to an arbitrary number of streams.

The computation model, called *mp2s-automata*[1], can be described as follows: Let $\mathbb{D}$ be a set, and let m, k_f, k_b be integers with $m \geqslant 1$ and $k_f, k_b \geqslant 0$. An

mp2s-automaton $\mathcal{A}$ with parameters $(\mathbb{D}, m, k_f, k_b)$

receives as input two streams $\vec{S} \in \mathbb{D}^*$ and $\vec{T} \in \mathbb{D}^*$. The automaton's memory consists of m different states (note that this corresponds to a memory buffer consisting of $\lg m$ bits). The automaton's state space is denoted by Q. We assume that Q contains a designated *start state* and that there is a designated subset F of Q of so-called *accepting states*.

On each of the input streams $\vec{S}$ and $\vec{T}$, the automaton has k_f heads that process the stream from left to right (so-called *forward heads*) and k_b heads that process the stream from right to left (so-called *backward heads*). The heads are allowed to move asynchronously. We use k to denote the total number of heads, i.e., $k = 2k_f + 2k_b$.

In the *initial configuration* of $\mathcal{A}$ on input $(\vec{S}, \vec{T})$, the automaton is in the *start state*, all *forward* heads on $\vec{S}$ and $\vec{T}$ are placed on the leftmost element in the stream, i.e., s_1 resp. t_1, and all *backward* heads are placed on the rightmost element in the stream, i.e., $s_{|\vec{S}|}$ resp. $t_{|\vec{T}|}$.

During each computation step, depending on (a) the current state (i.e., the current content of the automaton's memory) and (b) the elements of $\vec{S}$ and $\vec{T}$ at the current head positions, a deterministic transition function determines (1) the next state (i.e., the new content of the automaton's memory) and (2) which of the k heads should be advanced to the next position (where forward heads are

[1] *"mp2s"* stands for multi-pass processing of 2 streams

$\mathbb{D}$	:	set of *data items* of which input streams $\vec{S}$ and $\vec{T}$ are composed
m	:	size of the automaton's *state space* Q (this corresponds to $\lg m$ bits of memory)
k_f	:	number of *forward heads* available on each input stream
k_b	:	number of *backward heads* available on each input stream
k	:	$2k_f + 2k_b$ (total number of heads)

Figure 1: The meaning of the parameters $(\mathbb{D}, m, k_f, k_b)$ of an mp2s-automaton.

advanced one step to the right, and backward heads are advanced one step to the left). Formally, the transition function can be specified in a straightforward way by a function

$$\delta \;:\; Q \times (\mathbb{D} \cup \{\textit{end}\})^k \;\longrightarrow\; Q \times \{\textit{advance}, \textit{stay}\}^k$$

where Q denotes the automaton's state space, and *end* is a special symbol (not belonging to $\mathbb{D}$) which indicates that a head has reached the end of the stream (for a forward head this means that the head has been advanced beyond the rightmost element of the stream, and for a backward head this means that the head has been advanced beyond the leftmost element of the stream).

The automaton's computation on input $(\vec{S}, \vec{T})$ ends as soon as each head has passed the entire stream. The input is *accepted* if the automaton's state then belongs to the set F of accepting states, and it is *rejected* otherwise.

The computation model of mp2s-automata is closely related to the *finite cursor machines* of [14]. In both models, several streams can be processed in parallel, and several heads (or, "cursors") may be used to perform several "asynchronous" passes over the same stream in parallel. In contrast to the mp2s-automata of the present paper, finite cursor machines were introduced as an abstract model for database query processing, and their formal definition in [14] is presented in the framework of *abstract state machines*.

Note that mp2s-automata can be viewed as a generalization of other models for one-pass or multi-pass processing of streams. For example, the scenario of [28], where a single pass over two streams is performed, is captured by an mp2s-automaton where 1 forward head and no backward heads are available on each stream. Also, the scenario where p consecutive passes of each input stream are available (cf., e.g., [20]), can be implemented by an mp2s-automaton: just use p forward heads and 0 backward heads, and let the i-th head wait at the first position of the stream until the $(i-1)$-th head has reached the end of the stream.

4. The set disjointness problem

Throughout Section 4 we consider a particular version of the *set disjointness problem* where, for each integer $n \geqslant 1$, $\mathbb{D}_n := \{a_1, b_1, \ldots, a_n, b_n\}$ is a fixed set of $2n$ data items. We write $\textit{Disj}_n$ to denote the following decision problem: The input consists of two streams $\vec{S}$ and $\vec{T}$ over $\mathbb{D}_n$ with $|\vec{S}| = |\vec{T}| = n$. The goal is to decide whether the sets $\{s_1, \ldots, s_n\}$ and $\{t_1, \ldots, t_n\}$ are disjoint.

An mp2s-automaton *solves* the problem $\textit{Disj}_n$ if, for all valid inputs to $\textit{Disj}_n$ (i.e., all $\vec{S}, \vec{T} \in \mathbb{D}^*$ with $|\vec{S}| = |\vec{T}| = n$), it accepts the input if, and only if, the corresponding sets are disjoint.

4.1. Two upper bounds for the set disjointness problem

It is straightforward to see that the problem *Disj*$_n$ can be solved by an mp2s-automaton with 2^{2n} states and a single forward head on each of the two input streams: During a first phase, the head on $\vec{S}$ processes $\vec{S}$ and stores, in the automaton's current state, the subset of $\mathbb{D}_n$ that has been seen while processing $\vec{S}$. Afterwards, the head on $\vec{T}$ processes $\vec{T}$ and checks whether the element currently seen by this head belongs to the subset of $\mathbb{D}_n$ that is stored in the automaton's state. Clearly, 2^{2n} states suffice for this task, since $|\mathbb{D}_n| = 2n$. We thus obtain the following trivial upper bound:

Proposition 4.1. *Disj*$_n$ *can be solved by an mp2s-automaton with parameters* $(\mathbb{D}_n, 2^{2n}, 1, 0)$.

The following result shows that, at the expense of increasing the number of forward heads on each stream to $\sqrt{n}$, the memory consumption can be reduced exponentially:

Proposition 4.2. *Disj*$_n$ *can be solved by an mp2s-automaton with parameters* $(\mathbb{D}_n, n{+}2, \sqrt{n}, 0)$.[2]

Proof. The automaton proceeds in two phases.

The goal in *Phase 1* is to move, for each $i \in \{1, \ldots, \sqrt{n}\}$, the i-th head on $\vec{S}$ onto the $\big((i{-}1)\sqrt{n}\ +1\big)$-th position in $\vec{S}$. This way, after having finished *Phase 1*, the heads partition $\vec{S}$ into $\sqrt{n}$ sub-streams, each of which has length $\sqrt{n}$. Note that $n + 1 - \sqrt{n}$ states suffice for accomplishing this: The automaton simply stores, in its state, the current position of the rightmost head(s) on $\vec{S}$. It starts by leaving head 1 at position 1 and moving the remaining heads on $\vec{S}$ to the right until position $\sqrt{n} + 1$ is reached. Then, it leaves head 2 at position $\sqrt{n} + 1$ and proceeds by moving the remaining heads to the right until position $2\sqrt{n} + 1$ is reached, etc.

During *Phase 2*, the automaton checks whether the two sets are disjoint. This is done in $\sqrt{n}$ sub-phases. During the j-th sub-phase, the j-th head on $\vec{T}$ processes $\vec{T}$ from left to right and compares each element in $\vec{T}$ with the elements on the current positions of the $\sqrt{n}$ heads on $\vec{S}$. When the j-th head on $\vec{T}$ has reached the end of the stream, each of the heads on $\vec{S}$ is moved one step to the right. This finishes the j-th sub-phase. Note that *Phase 2* can be accomplished by using just 2 states: By looking at the combination of heads on $\vec{T}$ that have already passed the entire stream, the automaton can tell which sub-phase it is currently performing. Thus, for *Phase 2* we just need one state for indicating that the automaton is in *Phase 2*, and an additional state for storing that the automaton has discovered already that the two sets are *not* disjoint. ∎

4.2. Two lower bounds for the set disjointness problem

We first show a lower bound for mp2s-automata where only forward heads are available:

Theorem 4.3. *For all integers* n, m, k_f, *such that, for* $k = 2k_f$ *and* $v = k_f^2 + 1$,

$$k^2 \cdot v \cdot \lg(n{+}1) \ + \ k \cdot v \cdot \lg m \ + \ v \cdot (1 + \lg v) \ \leqslant \ n\,,$$

the problem Disj$_n$ *cannot be solved by any mp2s-automaton with parameters* $(\mathbb{D}_n, m, k_f, 0)$.

Proof. Let n, m, and k_f be chosen such that they meet the theorem's assumption. For contradiction, let us assume that $\mathcal{A}$ is an mp2s-automaton with parameters $(\mathbb{D}_n, m, k_f, 0)$ that solves the problem *Disj*$_n$.

[2] To be precise, the proof shows that already $n + 2 - \sqrt{n}$ states suffice.

Recall that $\mathbb{D}_n = \{ a_1, b_1, \ldots, a_n, b_n \}$ is a fixed set of $2n$ data items. Throughout the proof we will restrict attention to input streams $\vec{S}$ and $\vec{T}$ which are enumerations of the elements in a set

$$A^I \quad := \quad \{a_i : i \in I\} \cup \{b_i : i \in \overline{I}\}$$

for arbitrary $I \subseteq \{1, .., n\}$ and its complement $\overline{I} := \{1, .., n\} \setminus I$.
Note that for all $I_1, I_2 \subseteq \{1, .., n\}$ we have

$$A^{I_1} \text{ and } A^{I_2} \text{ are disjoint} \iff I_2 = \overline{I_1}. \tag{4.1}$$

For each $I \subseteq \{1, .., n\}$ we let $\vec{S}^I$ be the stream of length n which is defined as follows: For each $i \in I$, it carries data item a_i at position i; and for each $i \notin I$, it carries data item b_i at position i. The stream $\vec{T}^I$ contains the same data items as $\vec{S}^I$, but in the opposite order: For each $i \in I$, it carries data item a_i at position $n - i + 1$; and for each $i \notin I$, it carries data item b_i at position $n - i + 1$.

For sets $I_1, I_2 \subseteq \{1, .., n\}$, we write $D(I_1, I_2)$ to denote the input instance $\vec{S}^{I_1}$ and $\vec{T}^{I_2}$ for the problem *Disj*$_n$. From (4.1) and our assumption that the mp2s-automaton $\mathcal{A}$ solves *Disj*$_n$, we obtain that

$$\mathcal{A} \text{ accepts } D(I_1, I_2) \iff I_2 = \overline{I_1}. \tag{4.2}$$

Throughout the remainder of this proof, our goal is to find two sets $I, I' \subseteq \{1, .., n\}$ such that

(1) $I \neq I'$, and

(2) the accepting run of $\mathcal{A}$ on $D(I, \overline{I})$ is "similar" to the accepting run of $\mathcal{A}$ on $D(I', \overline{I'})$, so that the two runs can be combined into an accepting run of $\mathcal{A}$ on $D(I, \overline{I'})$ (later on in the proof, we will see what "similar" precisely means).

Then, however, the fact that $\mathcal{A}$ accepts input $D(I, \overline{I'})$ contradicts (4.2) and thus finishes the proof of Theorem 4.3.

For accomplishing this goal, we let

$$v \quad := \quad k_f^2 + 1 \tag{4.3}$$

be 1 plus the number of pairs of heads on the two streams. We subdivide the set $\{1, .., n\}$ into v consecutive blocks $B_1, \ldots, B_v$ of equal size $\frac{n}{v}$. I.e., for each $j \in \{1, .., v\}$, block B_j consists of the indices in $\{ (j-1)\frac{n}{v} + 1, \ldots, j\frac{n}{v} \}$.

We say that a pair (h_S, h_T) of heads of $\mathcal{A}$ *checks block* B_j during the run on input $D(I_1, I_2)$ if, and only if, at some point in time during the run, there exist $i, i' \in B_j$ such that head h_S is on element a_i or b_i in $\vec{S}^{I_1}$ and head h_T is on element $a_{i'}$ or $b_{i'}$ in $\vec{T}^{I_2}$.

Note that each pair of heads can check at most one block, since only forward heads are available and the data items in $\vec{T}^{I_2}$ are arranged in the reverse order (with respect to the indices i of elements a_i and b_i) than in $\vec{S}^{I_1}$. Since there are v blocks, but only $v - 1$ pairs (h_S, h_T) of heads on the two streams, we know that for each $I_1, I_2 \subseteq \{1, .., n\}$ there exists a block B_j that is *not checked* during $\mathcal{A}$'s run on $D(I_1, I_2)$.

In the following, we determine a set $X \subseteq \{I : I \subseteq \{1, .., n\}\}$ with $|X| \geqslant 2$ such that for all $I, I' \in X$, item (2) of our goal is satisfied. We start by using a simple averaging argument to find a $j_0 \in \{1, .., v\}$ and a set $X_0 \subseteq \{I : I \subseteq \{1, .., n\}\}$ such that

- for each $I \in X_0$, block B_{j_0} is not checked during $\mathcal{A}$'s run on input $D(I, \overline{I})$, and
- $|X_0| \geqslant \frac{2^n}{v}$.

For the remainder of the proof we fix $\hat{B} := B_{j_0}$.
We next choose a sufficiently large set $X_1 \subseteq X_0$ in which everything outside block $\hat{B}$ is fixed:
A simple averaging argument shows that there is a $X_1 \subseteq X_0$ and a $\hat{I} \subseteq \{1, \ldots, n\} \setminus \hat{B}$ such that

- for each $I \in X_1$, $I \setminus \hat{B} = \hat{I}$, and
- $|X_1| \geqslant \frac{|X_0|}{2^{n-\frac{n}{v}}} \geqslant 2^{\frac{n}{v} - \lg v}$.

We next identify a set $X_2 \subseteq X_1$ such that for all $I, I' \in X_2$ the runs of $\mathcal{A}$ on $D(I, \overline{I})$ and $D(I', \overline{I'})$ are "similar" in a sense suitable for item (2) of our goal. To this end, for each head h of $\mathcal{A}$ we let config_h^I be the *configuration* (i.e., the current state and the absolute positions of all the heads) in the run of $\mathcal{A}$ on input $D(I, \overline{I})$ at the particular point in time where head h has just left block $\hat{B}$ (i.e., head h has just left the last element a_i or b_i with $i \in \hat{B}$ that it can access). We let config^I be the ordered tuple of the configurations config_h^I for all heads h of $\mathcal{A}$. Note that the number of possible configurations config_h^I is $\leqslant m \cdot (n{+}1)^k$, since $\mathcal{A}$ has m states and since each of the $k = 2k_f$ heads can be at one out of $n{+}1$ possible positions in its input stream. Consequently, the number of possible k-tuples config^I of configurations is $\leqslant \left(m \cdot (n{+}1)^k\right)^k$.
A simple averaging argument thus yields a tuple c of configurations and a set $X_2 \subseteq X_1$ such that

- for all $I \in X_2$, $\mathit{config}^I = c$, and
- $|X_2| \geqslant \frac{|X_1|}{(m \cdot (n+1)^k)^k} \geqslant 2^{\frac{n}{v} - \lg v - k \lg m - k^2 \lg(n+1)}$.

Using the theorem's assumption on the numbers n, m, and k_f, one obtains that $|X_2| \geqslant 2$. Therefore, we can find two sets $I, I' \in X_2$ with $I \neq I'$.

To finish the proof of Theorem 4.3, it remains to show that the runs of $\mathcal{A}$ on $D(I, \overline{I})$ and on $D(I', \overline{I'})$ can be combined into a run of $\mathcal{A}$ on $D(I, \overline{I'})$ such that $\mathcal{A}$ (falsely) accepts input $D(I, \overline{I'})$. To this end let us summarize what we know about I and I' in X_2:

(a) I and I' only differ in block $\hat{B}$.

(b) Block $\hat{B}$ is not checked during $\mathcal{A}$'s runs on $D(I, \overline{I})$ and on $D(I', \overline{I'})$. I.e., while any head on $\vec{S}^I$ (resp. $\vec{S}^{I'}$) is at an element a_i or b_i with $i \in \hat{B}$, no head on $\vec{T}^{\overline{I}}$ (resp. $\vec{T}^{\overline{I'}}$) is on an element $a_{i'}$ or $b_{i'}$ with $i' \in \hat{B}$.

(c) Considering $\mathcal{A}$'s runs on $D(I, \overline{I})$ and on $D(I', \overline{I'})$, each time a head leaves the last position in $\hat{B}$ that it can access, both runs are are in exactly the same configuration. I.e., they are in the same state, and all heads are at the same absolute positions in their input streams.

Due to item (a), $\mathcal{A}$'s run on input $D(I, \overline{I'})$ starts in the same way as the runs on $D(I, \overline{I})$ and $D(I', \overline{I'})$: As long as no head has reached an element in block $\hat{B}$, the automaton has not yet seen any difference between $D(I, \overline{I'})$ on the one hand and $D(I, \overline{I})$ and $D(I', \overline{I'})$ on the other hand.

At some point in time, however, some head h will enter block $\hat{B}$, i.e., it will enter the first element a_i or b_i with $i \in \hat{B}$ that it can access. The situation then is as follows:

- If h is a head on $\vec{S}^I$, then, due to item (b), no head on $\vec{T}^{\overline{I'}}$ is at an element in $\hat{B}$. Therefore, until head h leaves block $\hat{B}$, $\mathcal{A}$ will go through the same sequence of configurations as in its run on input $D(I, \overline{I})$. Item (c) ensures that when h leaves block $\hat{B}$, $\mathcal{A}$ is in the same configuration as in its runs on $D(I, \overline{I})$ and on $D(I', \overline{I'})$.

- Similarly, if h is a head on $\vec{T}^{\overline{I'}}$, then, due to item (b), no head on $\vec{S}^I$ is at an element in $\hat{B}$. Therefore, until head h leaves block $\hat{B}$, $\mathcal{A}$ will go through the same sequence of configurations as in its run on input $D(I', \overline{I'})$. Item (c) ensures that when h leaves block $\hat{B}$, $\mathcal{A}$ is in the same configuration as in its runs on $D(I', \overline{I'})$ and on $D(I, \overline{I})$.

In summary, in $\mathcal{A}$'s run on $D(I, \overline{I'})$, each time a head h has just left the last element in block $\hat{B}$ that it can access, it is in exactly the same configuration as in $\mathcal{A}$'s runs on $D(I, \overline{I})$ and on $D(I', \overline{I'})$ at the points in time where head h has just left the last element in block $\hat{B}$ that it can access. After the last head has left block $\hat{B}$, $\mathcal{A}$'s run on $D(I, \overline{I'})$ finishes in exactly the same way as $\mathcal{A}$'s runs on $D(I, \overline{I})$ and $D(I', \overline{I'})$. In particular, it accepts $D(I, \overline{I'})$ (since it accepts $D(I, \overline{I})$ and $D(I', \overline{I'})$). This, however, is a contradiction to (4.2). Thus, the proof of Theorem 4.3 is complete. ∎

Remark 4.4. Let us compare the lower bound from Theorem 4.3 with the upper bound of Proposition 4.2: The upper bound tells us that *Disj*$_n$ can be solved by an mp2s-automaton with $n+2$ states and $\sqrt{n}$ forward heads on each input stream. The lower bound implies (for large enough n) that if just $\sqrt[5]{n}$ forward heads are available on each stream, not even $2^{\sqrt[3]{n}}$ states suffice for solving the problem *Disj*$_n$ with an mp2s-automaton.

Remark 4.5. A straightforward calculation shows that the assumptions of Theorem 4.3 are satisfied, for example, for all sufficiently large integers n and all integers m and k_f with $4k_f \leqslant \sqrt[4]{\frac{n}{\lg n}}$ and $\lg m \leqslant \frac{n}{4k_f \cdot (k_f^2+1)}$.

Theorem 4.3 can be generalized to the following lower bound for mp2s-automata where also backward heads are available:

Theorem 4.6. *For all n, m, k_f, k_b such that, for $k = 2k_f + 2k_b$ and $v = (k_f^2 + k_b^2 + 1) \cdot (2k_f k_b + 1)$,*

$$k^2 \cdot v \cdot \lg(n+1) \;+\; k \cdot v \cdot \lg m \;+\; v \cdot (1 + \lg v) \;\leqslant\; n\,,$$

the problem Disj$_n$ cannot be solved by any mp2s-automaton with parameters $(\mathbb{D}_n, m, k_f, k_b)$.

Proof. The overall structure of the proof is the same as in the proof of Theorem 4.3. We consider the same sets A^I, for all $I \subseteq \{1, \ldots, n\}$. The stream $\vec{S}^I$ is chosen in the same way as in the proof of Theorem 4.3, i.e., for each $i \in I$, the stream $\vec{S}^I$ carries data item a_i at position i; and for each $i \notin I$, it carries data item b_i at position i.

Similarly as in the proof of Theorem 4.3, the stream $\vec{T}^I$ contains the same data items as $\vec{S}^I$. Now, however, the order in which the elements occur in $\vec{T}^I$ is a bit more elaborate. For fixing this order, we choose the following parameters:

$$v_1 \;:=\; k_f^2 + k_b^2 + 1\,, \qquad v_2 \;:=\; 2k_f k_b + 1\,, \qquad v \;:=\; v_1 \cdot v_2\,. \tag{4.4}$$

We subdivide the set $\{1, \ldots, n\}$ into v_1 consecutive blocks $B_1, \ldots, B_v$ of equal size $\frac{n}{v_1}$. I.e., for each $j \in \{1, \ldots, v_1\}$, block B_j consists of the indices in $\{\, (j-1)\frac{n}{v_1} + 1,\, \ldots,\, j\frac{n}{v_1} \,\}$.
Afterwards, we further subdivide each block B_j into v_2 consecutive subblocks of equal size $\frac{n}{v}$. These subblocks are denoted $B_j^1, \ldots, B_j^{v_2}$. Thus, each subblock $B_j^{j'}$ consists of the indices in $\{\, (j-1)\frac{n}{v_1} + (j'-1)\frac{n}{v} + 1,\, \ldots,\, (j-1)\frac{n}{v_1} + j'\frac{n}{v} \,\}$.

Now let π be the permutation of $\{1, \ldots, n\}$ which maps, for all j, r with $1 \leqslant j \leqslant v_1$ and $1 \leqslant r \leqslant \frac{n}{v_1}$, element $(j-1)\frac{n}{v_1} + s$ onto element $(v_1 - j)\frac{n}{v_1} + s$. Thus, π maps elements in block B_j onto elements in block B_{v_1-j+1}, and inside these two blocks, π maps the elements of subblock

$B_j^{j'}$ onto elements in subblock $B_{v_1-1+1}^{j'}$. Note that π reverses the blocks B_j in order, but it does *not* reverse the order of the subblocks $B_j^{j'}$.

Finally, we are ready to fix the order in which the elements in A^I occur in the stream $\vec{T}^I$: For each $i \in I$, the stream $\vec{T}^I$ carries data item a_i at position $\pi(i)$; and for each $i \notin I$, it carries data item b_i at position $\pi(i)$.

In the same way as in the proof of Theorem 4.3, we write $D(I_1, I_2)$ to denote the input instance $\vec{S}^{I_1}$ and $\vec{T}^{I_2}$.

A pair of heads (h_S, h_T) is called *mixed* if one of the heads is a forward head and the other is a backward head. Since π reverses the order of the blocks $B_1, \ldots, B_{v_1}$, it is straightforward to see that every *non-mixed* pair of heads can check at most one of the blocks $B_1, \ldots, B_{v_1}$. Since there are v_1 blocks, but only $(v_1 - 1)$ non-mixed pairs of heads, we know that for all $I_1, I_2 \subseteq \{1, \ldots, n\}$ there exists a block B_j that is *not checked* by any non-mixed pair of heads during $\mathcal{A}$'s run on input $D(I_1, I_2)$.

The same averaging argument as in the proof of Theorem 4.3 thus tells us that there is a $j_1 \in \{1, \ldots, v_1\}$ and a set $X_0' \subseteq \{I : I \subseteq \{1, \ldots, n\}\}$ such that

- for each $I \in X_0'$, block B_{j_1} is not checked by any non-mixed pair of heads during $\mathcal{A}$'s run on input $D(I, \overline{I})$, and
- $|X_0'| \geqslant \frac{2^n}{v_1}$.

From our particular choice of π, it is straightforward to see that every *mixed* pair of heads can check at most one of the subblocks $B_{j_1}^1, \ldots, B_{j_1}^{v_2}$. Since there are v_2 such subblocks, but only $(v_2 - 1)$ mixed pairs of heads, there must be a $j_2 \in \{1, \ldots, v_2\}$ and a set $X_0 \subseteq X_0'$ such that

- for each $I \in X_0$, subblock $B_{j_1}^{j_2}$ is not checked by any pair of heads during $\mathcal{A}$'s run on input $D(I, \overline{I})$, and
- $|X_0| \geqslant \frac{|X_0'|}{v_2} \geqslant \frac{2^n}{v}$.

For the remainder of the proof we fix $\hat{B} := B_{j_1}^{j_2}$, and we let $k := 2k_f + 2k_b$ denote the total number of heads. Using these notations, the rest of the proof can be taken vertatim from the proof of Theorem 4.3. ∎

The proof of Theorem 4.6 is implicit in [14] (see Theorem 5.11 in [14]). There, however, the proof is formulated in the terminology of a different machine model, the so-called *finite cursor machines*.

5. Final remarks

Several questions concerning the computational power of mp2s-automata occur naturally. On a technical level, it would be nice to determine the exact complexity of the set disjointness problem with respect to mp2s-automata. In particular: Is the upper bound provided by Proposition 4.2 optimal? Can backward scans significantly help for solving the set disjointness problem? Are $\sqrt{n}$ heads really necessary for solving the set disjointness problem when only a sub-exponential number of states are available?

A more important task, however, is to consider also randomized versions of mp2s-automata, to design efficient randomized approximation algorithms for particular problems, and to develop techniques for proving lower bounds in the randomized model.

Acknowledgement. I would like to thank Georg Schnitger for helpful comments on an earlier version of this paper.

References

[1] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proc. FOCS'04*, pages 540–549, 2004.
[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58:137–147, 1999.
[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. PODS'02*, pages 1–16, 2002.
[4] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the memory requirements of XPath evaluation over XML streams. In *Proc. PODS'04*, pages 177–188, 2004.
[5] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *Proc. PODS'05*, pages 216–227, 2005.
[6] P. Beame and D.-T. Huynh-Ngoc. On the value of multiple read/write streams for approximating frequency moments. In *Proc. FOCS'08*, 2008.
[7] P. Beame, T. S. Jayram, and A. Rudra. Lower bounds for randomized read/write stream algorithms. In *Proc. STOC'07*, pages 689–698, 2007.
[8] C. Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. *VLDB Journal*, 11(4):354–379, 2002.
[9] T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.
[10] A. Czumaj and C. Sohler. Subliner-time algorithms. *Bulletin of the EATCS*, 89:23–47, 2006.
[11] C. Demetrescu, I. Finocchi, and A. Ribichini. Trading off space for passes in graph streaming problems. In *Proc. SODA'06*, pages 714–723, 2006.
[12] A. Gál and P. Gopalan. Lower bounds on streaming algorithms for approximating the length of the longest increasing subsequence. In *Proc. FOCS'07*, pages 294–304, 2007.
[13] T. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing XML streams with deterministic automata and stream indexes. *ACM Transactions on Database Systems*, 29(4):752–788, 2004.
[14] M. Grohe, Y. Gurevich, D. Leinders, N. Schweikardt, J. Tyszkiewicz, and J. Van den Bussche. Database query processing using finite cursor machines. *Theory of Computing Systems*, 2009. To appear. A preliminary version can be found in *Proc. ICDT'07*, pages 284–298.
[15] M. Grohe, A. Hernich, and N. Schweikardt. Randomized computations on large data sets: Tight lower bounds. In *Proc. PODS'06*, pages 243–252, 2006. Full version available as CoRR Report, arXiv:cs.DB/0703081.
[16] M. Grohe, C. Koch, and N. Schweikardt. Tight lower bounds for query processing on streaming and external memory data. Accepted at *Theoretical Computer Science*, special issue for selected papers from ICALP'05.
[17] M. Grohe and N. Schweikardt. Lower bounds for sorting with few random accesses to external memory. In *Proc. PODS'05*, pages 238–249, 2005.
[18] S. Guha and A. McGregor. Tight lower bounds for multi-pass stream computation via pass elimination. In *Proc. ICALP'08*, pages 760–772, 2008.
[19] Y. Gurevich, D. Leinders, and J. Van den Bussche. A theory of stream queries. In *Proc. DBPL*, pages 153–168, 2007.
[20] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In *External memory algorithms*, volume 50, pages 107–118. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1999.
[21] A. McGregor et al. Open problems in data streams and related topics, December 2006. IITK Workshop on Algorithms for Data Streams. `http://www.cse.iitk.ac.in/users/sganguly/workshop.html`.
[22] J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
[23] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.
[24] M. Ruhl. *Efficient Algorithms for New Computational Models.* PhD thesis, Massachusetts Institute of Technology, 2003.
[25] N. Schweikardt. Machine models and lower bounds for query processing. In *Proc. PODS'07*, pages 41–52, 2007.

[26] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *Proc. ICDT'07*, pages 299–313, 2007.
[27] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. PODS'02*, pages 53–64, 2002.
[28] M. Shalem and Z. Bar-Yossef. The space complexity of processing XML twig queries over indexed documents. In *Proc. ICDE'08*, pages 824–832, 2008.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 63–74
www.stacs-conf.org

SHORTEST PATHS AVOIDING FORBIDDEN SUBPATHS

MUSTAQ AHMED [1] AND ANNA LUBIW [1]

[1] David R. Cheriton School of Computer Science, University of Waterloo, Canada
E-mail address: m6ahmed@uwaterloo.ca
E-mail address: alubiw@uwaterloo.ca

Abstract. In this paper we study a variant of the shortest path problem in graphs: given a weighted graph G and vertices s and t, and given a set X of forbidden paths in G, find a shortest s-t path P such that no path in X is a subpath of P. Path P is allowed to repeat vertices and edges. We call each path in X an *exception*, and our desired path a *shortest exception avoiding path*. We formulate a new version of the problem where the algorithm has no a priori knowledge of X, and finds out about an exception $x \in X$ only when a path containing x fails. This situation arises in computing shortest paths in optical networks. We give an algorithm that finds a shortest exception avoiding path in time polynomial in $|G|$ and $|X|$. The main idea is to run Dijkstra's algorithm incrementally after replicating vertices when an exception is discovered.

1. Introduction

One of the most fundamental combinatorial optimization problems is that of finding shortest paths in graphs. In this paper we study a variant of the shortest path problem: given a weighted graph $G(V, E)$, and vertices s and t, and given a set X of *forbidden paths* in G, find a shortest s-t path P such that no path in X is a subpath of P. We call paths in X *exceptions*, and we call the desired path a *shortest exception avoiding path*. We allow an exception avoiding path to be non-simple, i.e., to repeat vertices and edges. In fact the problem becomes hard if the solution is restricted to simple paths [20]. This problem has been called the *Shortest Path Problem with Forbidden Paths* by Villeneuve and Desaulniers [22]. Unlike them, we assume no a priori knowledge of X. More precisely, we can identify a forbidden path only after failing in our attempt to follow that path. This variant of the problem has not been studied before. It models the computation of shortest paths in optical networks, described in more detail in the "Motivation" section below. Note that when we fail to follow a path because of a newly discovered exception, we are still interested in a shortest path from s to t as opposed to a detour from the failure point. This is what is required in optical networks, because intermediate nodes do not store packets, and hence s must resend any lost packet.

1998 ACM Subject Classification: G.2.2; F.2.2.
Key words and phrases: Algorithms and data structures; Graph algorithms; Optical networks.
Research partially supported by Nortel Networks and NSERC.

This paper presents two algorithms to compute shortest exception avoiding paths in the model where exceptions are not known a priori. The algorithms take respectively $O(kn \log n + km)$ and $O((n + L) \log(n + L) + m + dL)$ time to find shortest exception avoiding paths from s to all other vertices, where $n = |V|$, $m = |E|$, d is the largest degree of a vertex, k is the number of exceptions in X, and L is the total size of all exceptions.

Our algorithm uses a vertex replication technique similar to the one used to handle non-simple paths in other shortest path problems [6, 22]. The idea is to handle a forbidden path by replicating its vertices and judiciously deleting edges so that one copy of the forbidden path is missing its last edge and the other copy is missing its first edge. The result is to exclude the forbidden path but allow all of its subpaths. The main challenge is that vertex replication can result in an exponential number of copies of any forbidden path that overlaps the current one. Villeneuve and Desaulniers [22] address this challenge by identifying and compressing the overlaps of forbidden paths, an approach that is impossible for us since we do not have access to X. Our new idea is to couple vertex replication with the "growth" of a shortest path tree. By preserving certain structure in the shortest path tree we prove that the extra copies of forbidden paths that are produced during vertex replication are immaterial. Our algorithm is easy to implement, yet the proof of correctness and the run-time analysis are non-trivial.

1.1. Motivation

Our research on shortest exception avoiding path was motivated by a problem in optical network routing from Nortel Networks. In an optical network when a ray of light of a particular wavelength tries to follow a path P consisting of a sequence of optical fibers, it may fail to reach the endpoint of P because of various transmission impairments such as attenuation, crosstalk, dispersion and non-linearities [12, 17]. This failure may happen even though the ray is able to follow *any* subpath P' of P. This non-transitive behavior occurs because those impairments depend on numerous physical parameters of the traversed path (e.g., length of the path, type of fiber, wavelength and type of laser used, location and gain of amplifiers, number of switching points, loss per switching point, etc.), and the effect of those parameters may be drastically different in P than in P' [2]. Forbidden subpaths provide a straight-forward model of this situation.

We now turn to the issue of identifying forbidden paths. Because of the large number of physical parameters involved, and also because many of the parameters vary over the lifetime of the component [3], it is not easy to model the feasibility of a path. Researchers at Nortel suggested a model whereby an algorithm identifies a potential path, and then this path is tried out on the actual network. In case of failure, further tests can be done to pinpoint a minimal forbidden subpath. Because such tests are expensive, a routing algorithm should try out as few paths as possible. In particular it is practically impossible to identify all forbidden paths ahead of time—we have an exponential number of possible paths to examine in the network. This justifies our assumption of having no a priori knowledge of the forbidden paths, and of identifying forbidden paths only by testing feasibility of a path.

The shortest exception avoiding path problem may also have application in vehicle routing. Forbidden subpaths involving pairs of edges occur frequently ("No left turn") and can occur dynamically due to rush hour constraints, lane closures, construction, etc. Longer forbidden subpaths are less common, but can arise, for example if heavy traffic makes it impossible to turn left soon after entering a multi-lane roadway from the right. If we are

routing a single vehicle it is more natural to find a detour from the point of failure when a forbidden path is discovered. This is different from our model of rerouting from s upon discovery of a forbidden path. However, in the situation when vehicles will be dispatched repeatedly, our model does apply.

1.2. Preliminaries

We are given an directed graph $G(V, E)$ with $n = |V|$ vertices and $m = |E|$ edges where each edge $e \in E$ has a positive weight denoting its *length*. We are also given a source vertex $s \in V$, a destination vertex $t \in V$, and a set X of paths in G. The graph G together with X models a communication network in which a packet cannot follow any path in X because of the physical constraints mentioned in Sec. 1.1. We assume that the algorithm can access the set X of forbidden paths only by performing queries to an oracle. Each query is a path P, and the oracle's response is either the confirmation that P is exception avoiding, or else an exception $x \in X$ that is a subpath of P and whose last vertex is earliest in P. Ties can be broken arbitrarily. In our discussion we say "we try a path" instead of saying "we query the oracle" because the former is more intuitive. In Sec. 4 we modify our algorithm for the case of an oracle that returns *any* exception on a path (not just the one that ends earliest). This requires more calls to the oracle but gives a faster run-time.

We want to find a shortest path from s to t that does not contain any path in X as a subpath—we make the goal more precise as follows. A *path* is a sequence of vertices each joined by an edge to the next vertex in the sequence. Note that we allow a path to visit vertices and edges more than once. If a path does not visit any vertex more than once, we explicitly call it a *simple path*. A simple directed path from vertex v to vertex w in G is called a *forbidden path* or an *exception* if a packet cannot follow the path from v to w because of the physical constraints. Given a set A of forbidden paths, a path $(v_1, v_2, v_3, \ldots, v_l)$ is said to *avoid* A if $(v_i, v_{i+1}, \ldots v_j) \notin A$ for all i, j such that $1 \leq i < j \leq l$. A path P from s to t is called a *shortest A-avoiding path* if the length of P is the shortest among all A-avoiding paths from s to t. We will use the term "exception avoiding" instead of "X-avoiding" when A is equal to X, the set of all forbidden paths in G.

1.3. Related work

A shortest s-t path in a graph can be computed in $O(n \log n + m)$ time and linear space using Dijkstra's algorithm with Fibonacci heaps if all edge weights are non-negative, and in $O(mn)$ time and linear space using the Bellman-Ford algorithm otherwise [5]. When the edge weights are non-negative integers, the problem can be solved in deterministic $O(m \log\log n \log\log\log n)$ time and linear space if the graph is directed [13], and in optimal $O(m)$ time if the graph is undirected [21]. In many of these cases, there are randomized algorithms with better expected times as well as approximation schemes. See Zwick [23] for a survey of shortest path algorithms, and Cabello [4], Goldberg and Harrelson [11] and Holzer et al. [15] for some of the more recent work.

Two recent papers on shortest paths in graphs address the issue of avoiding a set of forbidden paths, assuming that all the forbidden paths are known a priori. The first paper gives a hardness result. Szeider [20] shows, using a reduction from 3-SAT, that the problem of finding a shortest *simple* exception avoiding path is NP-complete even when each forbidden path has two edges. If the forbidden paths are *not* known a priori, the

hardness result still applies to the case of simple paths because the lack of prior knowledge of the forbidden paths only makes the problem harder.

The second paper, by Villeneuve and Desaulniers [22], gives an algorithm for a shortest (possibly non-simple) exception avoiding path for the case when all the forbidden paths are known a priori. They preprocess the graph in $O((n+L)\log(n+L)+m+dL)$ time and $O(n+m+dL)$ space so that a shortest path from s to a query vertex can be found in $O(n+L)$ time. They first build a deterministic finite automaton (DFA) from the set of forbidden paths using the idea of Aho and Corasick [1], which can detect in linear time whether a given path contains any of the forbidden paths. They then "insert" the DFA into G by replicating certain vertices of G in the manner introduced by Martins [6], and then build a shortest path tree in this modified graph. Their algorithm cannot handle the case where the set of all forbidden paths is not explicitly given. Our algorithm is strictly more general, and we show in Sec. 4 that it solves their problem in roughly the same time but in less ($O(n+m+L)$) space.

We now mention two problems that seem related to ours, but do not in fact provide solutions to ours. The first one is maintaining shortest paths in a dynamic graph, i.e., where nodes or edges may fail [7, 9, 14], or edge weights may change (e.g., [7, 8]). Forbidden paths cannot be modeled by deleting edges or by modifying edge costs because *all* edges in a particular forbidden path may be essential—see Fig. 1 for an example. The second seemingly related problem is finding the k shortest paths in a graph. This was the subject of Martins [6] who introduced the vertex replication technique that we use in our algorithm. There is considerable work on this problem, see Eppstein [10] for a brief survey. But the k shortest path problem is again different from our situation because a forbidden subpath may be a bottleneck that is present in all of the k shortest paths even for $k \in \Omega(2^{n/2})$, see Villeneuve and Desaulniers [22].

In the context of optical networks researchers have studied many theoretical problems. See Ramaswami and Sivarajan [19] for details on optical networks, and Lee and Shayman [17] and McGregor and Shepherd [18] for a brief survey of the theoretical problems that have been investigated. In the previous work, the effect of physical constraints on paths in optical networks is either not considered at all (e.g., Khuller et al. [16]), or simply modeled by a known constant upper bound on the length of such a path (e.g., Gouveia et al [12], Lee and Shayman [17] and McGregor and Shepherd [18]). To the best of our knowledge, none of the previous work on shortest paths in optical networks considers the fact that it is practically infeasible to know a priori all the forbidden paths in the network, i.e., all the constraints in X. Our paper handles the issue of physical constraints from a different and much more practical perspective.

2. Algorithm for a shortest s-t path

In our algorithm we begin with a shortest path tree rooted at s, ignoring the exceptions. We then "try out" the path from s to t in the tree. If the path is free of exceptions, we are done. Otherwise, to take the newly discovered exception into account, we modify the graph using path replication as described in the Introduction, and we modify the shortest path tree to match. In general, we maintain a modified graph and a shortest path tree in the graph that gives a shortest path in the original graph from s to every other vertex avoiding all the currently-known exceptions. We will first illustrate the idea with an example. Consider the graph G in Fig. 1(a), where the integers denote edge weights, and the dashed arrow marks

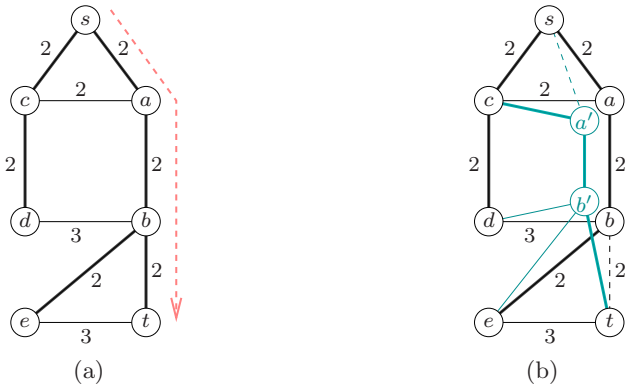
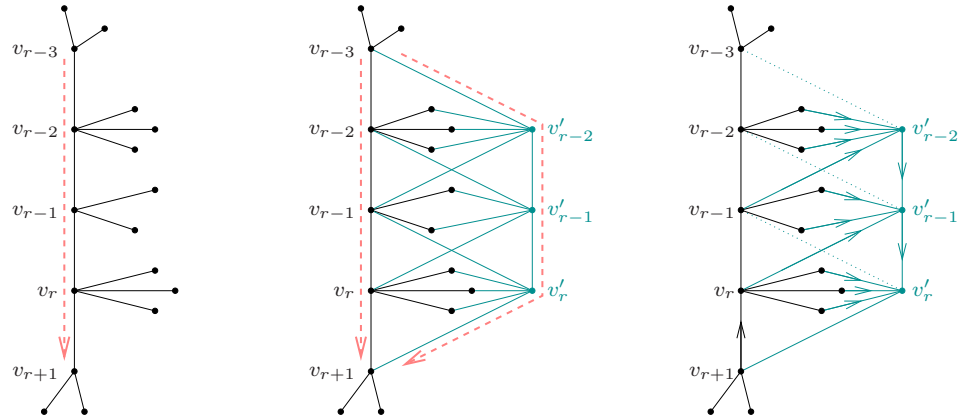


Figure 1: (a) Shortest paths and (b) shortest x-avoiding paths in a graph, where $x = (s, a, b, t)$.

the forbidden path $x = (s, a, b, t)$. Note that for simplicity we have used undirected edges in the figure to denote bidirectional edges. It is not hard to see that $P = (s, c, a, b, t)$ is the shortest x-avoiding path from s to t. To find P, we first construct a shortest path tree rooted at s (marked using the heavy edges in Fig. 1(a)), and then try the path (s, a, b, t) in the tree. The path fails because it contains x, so we use a *vertex replication technique* similar to the one by Martins [6] to make duplicates of vertices a and b and delete edges (s, a') and (b, t), as shown in Fig. 1(b). We then construct a shortest path tree rooted at s (marked using the heavy edges in Fig. 1(b)) in the modified graph, and try the path (s, c, a', b', t) which "represents" the path P in G. We are done if x is the only forbidden path in G. Note that this approach can double the number of *undiscovered* forbidden paths. Suppose $y = (c, a, b)$ is another forbidden path in G. We have two copies of y in the modified graph: (c, a, b) and (c, a', b'), and we have to avoid both of them. Our solution to this doubling problem is to "grow" the shortest path tree in such a way that at most one of these two copies is encountered in future. Our algorithm is as follows:

```
1 construct the shortest path tree T_0 rooted at s in G_0 = G;
2 let i = 1;
3 send a packet from s to t through the path in T_0;
4 while the packet fails to reach t do
5     let x_i be the exception that caused the failure;
6     construct G_i from G_{i−1} by replicating the intermediate vertices of x_i and then
      deleting selected edges;
7     construct the shortest path tree T_i rooted at s in G_i using T_{i−1};
8     send a packet from s to t through the path in T_i;
9     let i = i + 1;
```

In the above algorithm, the only lines that need further discussion are Lines 6 and 7; details are in Sections 2.1 and 2.2 respectively. In the rest of the paper, whenever we focus on a particular iteration $i > 0$, we use the following notation: (i) the path from s to t in T_{i-1}, i.e., the path along which we try to send the packet to t in Line 4 in the iteration, is $(s, v_1, v_2, \ldots, v_p, t)$, and (ii) the exception that prevented the packet from reaching t in the iteration is $x_i = (v_{r-l}, v_{r-l+1}, \ldots, v_r, v_{r+1})$, which consists of $l + 1$ edges.



(a) G_{i-1} (b) G'_{i-1} (c) G_i

Figure 2: Modifying G_{i-1} to G_i: (a) The part of G_{i-1} at an exception $(v_{r-3}, v_{r-2}, v_{r-1}, v_r, v_{r+1})$, with $l = 3$. (b) Replicating vertices to create G'_{i-1}. The dashed paths show two of the 8 copies of the exception. (c) Deleting edges to create G_i. The dotted lines denote deleted edges.

2.1. Modifying the graph

The modification of G_{i-1} into G_i (Line 6) in the ith iteration eliminates exception x_i while preserving all the x_i-avoiding paths in G_{i-1}. We do the modification in two steps.

In the first step, we create a graph G'_{i-1} by replicating the intermediate vertices of x_i (i.e., the vertices $v_{r-l+1}, v_{r-l+2}, \ldots, v_r$). We also add appropriate edges to the replica v' of a vertex v. Specifically, when we add v' to G_{i-1}, we also add the edges of appropriate weights between v' and the neighbors of v. It is easy to see that if a path in G_{i-1} uses $l' \leq l$ intermediate vertices of x_i, then there are exactly $2^{l'}$ copies of the path in G'_{i-1}. We say that a path in G'_{i-1} is x_i-avoiding if it contains none of the 2^l copies of x_i.

In the second step, we build a spanning subgraph G_i of G'_{i-1} by deleting a few edges from G'_{i-1} in such a way that all copies of x_i in G'_{i-1} are eliminated, but all x_i-avoiding paths in G'_{i-1} remain unchanged. To build G_i from G'_{i-1}, we delete the edges (v_{j-1}, v'_j) and (v'_j, v_{j-1}) for all $j \in [r-l+1, r]$. We also delete the edge (v_r, v_{r+1}), all the outgoing edges from v'_r except (v'_r, v_{r+1}), and all the outgoing edges from v'_j except (v'_j, v'_{j+1}) for all $j \in [r-l+1, r-1]$. Figure 2 shows how the "neighborhood" of an exception changes from G_{i-1} to G_i. As before, the undirected edges in the figure are bidirectional.

Observation 2.1. Graph G_i has no copy of x_i.

In Sec. 3.1 we will prove that G_i still contains all the x_i-avoiding paths of G_{i-1}.

The vertices in G_i $[G'_{i-1}]$ that exist also in G_{i-1} (i.e., the ones that are not replica vertices) are called the *old vertices of* G_i *[respectively* G'_{i-1}*]*. Note that the vertices of G_0 exist in G_i for all $i \geq 0$. These vertices are called the *original vertices of* G_i.

2.2. Constructing the tree

In Line 7 of our algorithm we construct a tree T_i that contains a shortest x_i-avoiding path from s to every other vertex in G_{i-1}. Tree T_i is rooted at s, and its edges are directed

away from s. Not every shortest path tree rooted at s in G_i will work. In order to guarantee termination of the algorithm, T_i must be similar to T_{i-1}, specifically, every x_i-avoiding path from s in T_{i-1} must be present in T_i. The necessity of this restriction is explained in Sec. 3.3.

We construct the required T_i by preserving as much of T_{i-1} as possible. We apply Dijkstra's algorithm starting from the part of T_{i-1} that can be preserved. Let V' be the set of vertices that are either replica vertices in G_i, or descendants of v_{r+1} in T_{i-1}. We first set the weight of each $v \in V'$ to infinity, and temporarily set $T_i = T_{i-1} - V'$. Then, for each $v \in V'$, we set the weight of v to the minimum, over all edges (u, v), of the sum of the weight of u and the length of (u, v). Finally, we initialize the queue used in Dijkstra with all the vertices in V' and run the main loop of Dijkstra's algorithm. Each iteration of the loop adds one vertex in V' to the temporary T_i. When the queue becomes empty, we get the final tree T_i.

3. Correctness and analysis

3.1. Justifying the graph modification

In this section we prove the following lemma, which uses the notion of a *corresponding path*. Consider any path P_i in G_i. By substituting every vertex in P_i that is not present in G_{i-1} with the corresponding old vertex in G_{i-1}, we get the corresponding path P_{i-1} in G_{i-1}. This is possible because any "new" edge in G_i is a replica of an edge in G_{i-1}. We define the corresponding path P_j in G_j for all $j < i$ by repeating this argument.

Lemma. If P_i is a shortest path from s to an original vertex v in G_i, P_0 is a shortest $\{x_1, x_2, \ldots, x_i\}$-avoiding path from s to v in G_0.

To prove the above lemma (repeated as Lemma 3.3 below), we will first prove that x_i-avoiding paths in G_{i-1} are preserved in G_i (Lemma 3.2), using the following characteristic of an x_i-avoiding path in the intermediate graph G'_{i-1}:

Lemma 3.1. *For any x_i-avoiding path P from s to v that uses only the old vertices in G'_{i-1}, there exists a copy of P in G_i that starts and ends at the old vertices s and v respectively, and possibly passes through the corresponding replicas of its intermediate vertices.*

Proof. Graph G_i contains all the edges between pairs of old vertices in G'_{i-1} except for the directed edge (v_r, v_{r+1}). Thus P can remain unchanged if it does not use this directed edge. Otherwise we will re-route any portion of P that uses the directed edge (v_r, v_{r+1}) to use the replica edge (v'_r, v_{r+1}) instead. Let $P = (s = w_1, w_2, \ldots, w_q = v)$, and (w_j, w_{j+1}) be an occurrence of (v_r, v_{r+1}) in P. Tracing P backwards from w_j, let $h \leq j$ be the minimum index such that $(w_h, w_{h+1}, \ldots, w_{j+1})$ is a subpath of x_i. Because P is x_i-avoiding, w_h must be an intermediate vertex of x_i. This implies that $h > 1$, since $s = w_1$ is not an intermediate vertex of x_i because of the following reasons: (i) x_i is a path in the shortest path tree rooted at s in G_i, and (ii) there is no replica of s in G_i. Therefore w_{h-1} exists. We will reroute the portion of P between w_{h-1} and w_{j+1} by using the corresponding replica vertices in place of the subpath $(w_h, \ldots, w_j)$ of x_i. Note that the required edges exist in G_i (since P does not contain the whole exception x_i), and that the portions of P that we re-route are disjoint along P. Moreover, P starts and ends at the old vertices s and v respectively. ∎

Lemma 3.2. *Any x_i-avoiding path from s to v in G_{i-1} has a copy in G_i that starts and ends at the old vertices s and v respectively, and possibly goes through the corresponding replicas of its intermediate vertices.*

Proof. Let P be the x_i-avoiding path in G_{i-1}. As we do not delete any edge to construct G'_{i-1} from G_{i-1}, P remains unchanged in G'_{i-1}. Moreover, P uses no replica vertex in G'_{i-1}. So, Lemma 3.1 implies that P exists in G_i with the same old vertices at the endpoints, possibly going through the corresponding replicas of the intermediate vertices. ∎

Lemma 3.3. *If P_i is a shortest path from s to an original vertex v in G_i, P_0 is a shortest $\{x_1, x_2, \ldots, x_i\}$-avoiding path from s to v in G_0.*

Proof. For any $j \in [0, i]$, let $X_j = \{x_{j+1}, x_{j+2}, \ldots, x_i\}$. We show that for any j, if P_j is a shortest X_j-avoiding path in G_j, then P_{j-1} is a shortest X_{j-1}-avoiding path in G_{j-1}. The lemma then follows by induction on j, with basis $j = i$, because $X_i = \emptyset$ and thus P_i is a shortest X_i-avoiding path in G_i.

If P_j is a shortest X_j-avoiding path in G_j, P_j is X_{j-1}-avoiding because P_j is x_j-avoiding by Observation 2.1, and $X_j \cup \{x_j\} = X_{j-1}$. So, the corresponding path P_{j-1} is also X_{j-1}-avoiding. If we assume by contradiction that P_{j-1} is not a shortest X_{j-1}-avoiding path in G_{j-1}, then there exists another path P'_{j-1} from s to v in G_{j-1} which is X_{j-1}-avoiding and is shorter than P_{j-1}. Since $x_j \in X_{j-1}$, P'_{j-1} is x_j-avoiding, and hence by Lemma 3.2, there is a copy P'_j of path P'_{j-1} in G_j which has the same original vertices at the endpoints. As P'_{j-1} is X_j-avoiding, P'_j is also X_j-avoiding. This is impossible because P'_j is shorter than P_j. Therefore, P_{j-1} is a shortest X_{j-1}-avoiding path in G_{j-1}. ∎

3.2. Justifying the tree construction

To show that the "incremental" approach used in Sec. 2.2 to construct T_i is correct, we first show that the part of T_{i-1} that we keep unchanged in T_i is composed of shortest paths in G_i:

Lemma 3.4. *For every vertex v that is not a descendant of v_{r+1} in T_{i-1}, the path P from s to v in T_{i-1} is a shortest path in G_i.*

Proof. First we show that P exists in G_i. Every vertex in T_{i-1} exists in G_i as an old vertex. So, P exists in G_i through the old vertices if no edge of P gets deleted in G_i. The only edge between a pair of old vertices in G_{i-1} that gets deleted in G_i is (v_r, v_{r+1}). Since v is not a descendant of v_{r+1} in T_{i-1}, P does not use the edge (v_r, v_{r+1}). Therefore, no edge of P gets deleted in G_i. So, P exists in G_i through the old vertices.

Neither the modification from G_{i-1} to G'_{i-1} nor the one from G'_{i-1} to G_i creates any "shortcut" between any pair of vertices. So, there is no way that the distance between a pair of old vertices decreases after these modifications. Since these modifications do not change P, which is a shortest path in G_{i-1}, P is a shortest path in G_i. ∎

Lemma 3.5. *The tree T_i is a shortest path tree in G_i.*

Proof. For every vertex v that is not a descendant of v_{r+1} in T_{i-1}, the path P from s to v in T_i is the same as the one in T_{i-1} and hence, a shortest path in G_i (Lemma 3.4). For all other vertices v in G_i, it follows from Dijkstra's algorithm that the path from s to v in T_i is a shortest path. ∎

Lemmas 3.3 and 3.5 together prove that our algorithm is correct provided it terminates, which we establish in the next section.

3.3. Analyzing time and space requirement

Although in every iteration we eliminate one exception by modifying the graph, we introduce copies of certain other exceptions through vertex replication. Still our algorithm does not iterate indefinitely because, as we will show in this section, the incremental construction of the shortest path tree (Sec. 2.2) guarantees that we do not discover more than one copy of any exception. We first show that any exception in G_{i-1} has at most two copies in G_i (Lemma 3.6), and then prove that one of these two copies is never discovered in the future (Lemma 3.7):

Lemma 3.6. *Let $y \neq x_i$ be any exception in G_{i-1}. If the last vertex of y is* not *an intermediate vertex of x_i, then G_i contains exactly one copy of y. Otherwise, G_i contains exactly two copies of y. In the latter case, one copy of y in G_i ends at the old vertex v and the other copy ends at the corresponding replica v'.*

Proof. Let $\pi = (w_1, w_2, \ldots, w_j)$ be a maximal sequence of vertices in y that is a subsequence of $(v_{r-l+1}, v_{r-l+2}, \ldots, v_r)$. Let w'_j be the replica of w_j in G_i. We will first show that if there is a vertex v in y right after π, then exactly one of the edges (w_j, v) and (w'_j, v) exists in G_i. Consider the subgraph of G_i induced on the set of replica vertices $\{v'_{r-l+1}, v'_{r-l+2}, \ldots, v'_r\}$: this subgraph is a directed path from v'_{r-l+1} to v'_r, and the only edge that goes out of this subgraph is (v'_r, v_{r+1}). Therefore, (i) when $(w_j, v) = (v_r, v_{r+1})$, $(w'_j, v) \in G_i$ and $(w_j, v) \notin G_i$, and (ii) otherwise, $(w_j, v) \in G_i$ and $(w'_j, v) \notin G_i$.

Now G_i has exactly two copies of π: one through the old vertices, and another through the replicas. The above claim implies that when there is a vertex v in y right after π, G_i has at most one copy of the part of y from w_1 to v. However, when π is a suffix of y, G_i has both the copies of the part of y from w_1 to w_j. The lemma then follows because any part of y that contains no intermediate vertex of x_i has exactly one copy in G_i. ∎

Lemma 3.7. *Let $y \neq x_i$ be any exception in G_{i-1} such that the last vertex of y is an intermediate vertex v of x_i. The copy of y that ends at the old vertex v in G_i is not discovered by the algorithm in any future iteration.*

Proof. The copy of the path $(s, v_1, v_2, \ldots, v_r)$ through the old vertices in G_i contains v. Let P be the part of this path from s to v. Clearly, $P \in T_{i-1}$, and P does not contain any exception because the oracle returns the exception with the earlier last vertex. So, the way we construct T_j from T_{j-1} for any iteration $j \geq i$ ensures that $P \in T_j$.

Let y_1 be the copy of y that ends at v. Now y_1 is not a subpath of P because P does not contain any exception. For any $j \geq i$, $P \in T_j$, and both P and y_1 end at the same vertex, therefore $y_1 \notin T_j$. So, a packet in iteration j will not follow y_1, and y_1 will not be discovered in that iteration. ∎

Lemma 3.8. *The* ***while*** *loop iterates at most $k = |X|$ times.*

Proof. For any iteration i, G_{i-1} contains x_i, and G_i does not contain x_i. Every exception other than x_i in G_{i-1} has either one or two copies in G_i (Lemma 3.6). By Lemma 3.7, if an exception has two copies in G_i, only one of them is relevant in the future. Thus the number of exceptions effectively decreases by one in each iteration. The lemma then follows. ∎

To determine the running time, observe that the number of vertices increases in each iteration. However, we run Dijkstra's algorithm on at most n vertices in any iteration, because the number of replica vertices added in each iteration is always less than the number of vertices in the part of the shortest path tree that is carried over from the previous tree in our incremental use of Dijkstra. Moreover, we can make sure that Dijkstra's algorithm examines at most m edges in iteration i, by deleting a few more edges from G_i after performing the graph modification described in Sec. 2.1. More precisely, for each old vertex $v \in \{v_{r-l+1}, v_{r-l+2}, \ldots, v_r\}$, since the label (i.e., the "distance" from s) put on v by Dijkstra's algorithm in the previous iterations remains unchanged later on, we can safely delete from G_i all the *incoming* edges of v without affecting future modifications. (Note that for all $j \in [r-l+1, r]$, old vertices v_j and v_{j+1} are no longer adjacent in G_i, although the edge (v_j, v_{j+1}) still exists in T_i.) It is not hard to see that the number of new edges in G_i is now equal to the number of edges deleted from G_{i-1}.

Theorem 3.9. *The algorithm computes a shortest X-avoiding path in $O(kn \log n + km)$ time and $O(n + m + L)$ space.*

Proof. The correctness of the algorithm follows from Lemmas 3.3 and 3.5.

Let l_i be the number of intermediate vertices of the exception discovered at the ith iteration (thus the size of the exception is $l_i + 2$). The ith iteration adds l_i vertices. Since the algorithm iterates k times (Lemma 3.8), there are $n + \sum_{i=1}^{k} l_i < n + L$ vertices in the graph at termination. Because in each iteration the number of added edges is equal to the number of deleted edges, the space requirement is $O(n + m + L)$.

Each iteration of our algorithm takes $O(|V| \log |V| + |E|) = O(n \log n + m)$ time, and the total time requirement follows. ∎

We note that in practice, the algorithm will not discover all k of the forbidden paths. It will discover only the ones that "interfere" in getting from s to t.

4. Extensions

This section contains: (1) an algorithm to compute shortest paths from s to every other vertex in G; (2) an analysis in the case when X is given explicitly; and (3) a version of the algorithm where the oracle returns *any* exception on a query path, rather than the exception that ends earliest.

The algorithm in Sec. 2 can be extended easily to compute a shortest path from s to every other vertex in G. We simply repeat the previous algorithm for every vertex in G, but with a small change: in every iteration (except of course the first one) we use the graph and the shortest path tree constructed at the end of previous iteration. Since every exception in X is handled at most once, the **while** loop still iterates at most k times, and therefore, the time and space requirements remain the same.

Theorem 4.1. *The algorithm computes shortest X-avoiding paths from s to all other vertices in $O(kn \log n + km)$ time and $O(n + m + L)$ space.*

Our algorithm applies when X is known explicitly; taking into account the cost of sorting X so that we can efficiently query whether a path contains an exception we obtain:

Theorem 4.2. *When X is known a priori, we can preprocess the graph in $O(kn \log(kn) + km)$ time and $O(n + m + L)$ space so that we can find a shortest X-avoiding path from s to any vertex in $O(n + L)$ time.*

Recall that Villeneuve and Desaulniers [22] solved this problem in $O((n+L)\log(n+L)+m+dL)$ preprocessing time, $O(n+m+dL)$ space and $O(n+L)$ query time. Our algorithm is more space efficient than theirs. Our preprocessing is slightly slower in general, although it is slightly faster in the special case $L=\Theta(kn)$ and $m=o(dn)$ (intuitively, when the exceptions are long, and the average degree of a vertex is much smaller than the largest degree).

Finally, returning to the case where X is not known a priori, we consider a weaker oracle that returns any exception on the query path, rather than the exception that ends earliest. At the cost of querying the oracle more often, we obtain a better run-time. The idea is to query the oracle *during* the construction of a shortest path tree. The algorithm is very similar to Dijkstra's, the only difference is that it handles exceptions inside Dijkstra's loop. More precisely, right after a vertex v is dequeued and added to the current tree, we try the s-v path in the tree. If the path is exception avoiding, we update the distances of the neighbors of v and go to the next iteration, as in "traditional" Dijkstra's algorithm. Otherwise, we remove v from the current tree, perform vertex replication and edge deletion as described in Sec. 2.1, and then go to the next iteration.

Theorem 4.3. *The algorithm described above computes shortest X-avoiding paths from s to all other vertices in $O((n+L)\log(n+L)+m+dL)$ time and $O(n+m+L)$ space.*

Proof. There are at most $n+L$ vertices in the modified graph in any iteration. So, the loop in the modified Dijkstra's algorithm executes at most $n+L$ times, and the priority queue holds at most $n+L$ entries. Moreover, within Dijkstra's loop vertex replication and edge deletion take $O(dL)$ time in total. The running time then follows. The proof of correctness is similar to that of Theorem 4.1 except that the "current" shortest path tree is no longer a spanning tree in the current graph. ∎

This new algorithm is faster than the old algorithm of Theorem 4.1 in general but makes as many as $n+L$ queries to the oracle versus at most k oracle queries for the old algorithm. The old algorithm is slightly faster in the special case $L=\Theta(kn)$ and $m=o(dn)$.

5. Conclusion

Motivated by the practical problem of finding shortest paths in optical networks, we introduced a novel version of the shortest path problem where we must avoid forbidden paths, but we only discover the forbidden paths by trying them. We gave an easily implementable, polynomial time algorithm that uses vertex replication and incremental Dijkstra.

As we have mentioned before, in practice our algorithms will not discover all the forbidden paths in X. In fact, the running time of each of our algorithms is determined by only the forbidden paths that "interfere" in getting from s to t. An interesting open problem is to bound the number of such paths. We conjecture that in a real optical network, the number of such paths is $o(k)$, and therefore, our algorithms run much faster in practice.

Acknowledgment

We wish to thank Erik Demaine for useful discussion, and an anonymous referess for important suggestions.

References

[1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[2] Peter Ashwood-Smith. Personal communication, 2007.

[3] Peter Ashwood-Smith, Don Fedyk, and Vik Saxena. Link viability constraints requirements for GMPLS-enabled networks. http://tools.ietf.org/html/draft-ashwood-ccamp-gmpls-constraint-reqts-00, July 2005. Internet draft, work in progress.

[4] Sergio Cabello. Many distances in planar graphs. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1213–1220, New York, NY, USA, 2006.

[5] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[6] Ernesto de Queiros Vieira Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18(1):123–130, October 1984.

[7] Camil Demetrescu, Stefano Emiliozzi, and Giuseppe F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 369–378, Philadelphia, PA, USA, 2004.

[8] Camil Demetrescu, Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. Maintaining shortest paths in digraphs with arbitrary arc weights: an experimental study. In *Proceedings of the Fourth International Workshop on Algorithm Engineering*, pages 218–229, London, UK, 2001.

[9] Camil Demetrescu, Mikkel Thorup, Rezaul A. Chowdhury, and Vijaya Ramachandran. Oracles for distances avoiding a failed node or link. *SIAM J. Comput.*, 37(5):1299–1318, 2008.

[10] David Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1999.

[11] Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A^* search meets graph theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 156–165, Philadelphia, PA, USA, 2005.

[12] Luis Gouveia, Pedro Patrício, Amaro de Sousa, and Rui Valadas. MPLS over WDM network design with packet level QoS constraints based on ILP models. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, April 2003.

[13] Yijie Han. Improved fast integer sorting in linear space. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 793–796, Philadelphia, PA, USA, 2001.

[14] John Hershberger, Subhash Suri, and Amit Bhosle. On the difficulty of some shortest path problems. *ACM Trans. Algorithms*, 3(1):5, 2007.

[15] Martin Holzer, Frank Schulz, Dorothea Wagner, and Thomas Willhalm. Combining speed-up techniques for shortest-path computations. *J. Exp. Algorithmics*, 10:2.5, 2005.

[16] Samir Khuller, Kwangil Lee, and Mark A. Shayman. On degree constrained shortest paths. In *Proceedings of the 13th Annual European Symposium on Algorithms*, pages 259–270, 2005.

[17] Kwangil Lee and Mark A. Shayman. Optical network design with optical constraints in IP/WDM networks. *IEICE Transactions on Communications*, E88-B(5):1898–1905, 2005.

[18] Andrew McGregor and Bruce Shepherd. Island hopping and path colouring with applications to WDM network design. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 864–873, Philadelphia, PA, USA, January 2007.

[19] Rajiv Ramaswami and Kumar N. Sivarajan. *Optical Networks: A Practical Perspective*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[20] Stefan Szeider. Finding paths in graphs avoiding forbidden transitions. *Discrete Appl. Math.*, 126(2-3):261–273, 2003.

[21] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999.

[22] Daniel Villeneuve and Guy Desaulniers. The shortest path problem with forbidden paths. *European Journal of Operational Research*, 165(1):97–107, 2005.

[23] Uri Zwick. Exact and approximate distances in graphs—a survey. In *Proceedings of the Ninth Annual European Symposium on Algorithms*, pages 33–48, London, UK, 2001.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 75–86
www.stacs-conf.org

GENERATING SHORTER BASES FOR HARD RANDOM LATTICES

JOËL ALWEN [1] AND CHRIS PEIKERT [2]

[1] New York University, 251 Mercer St., New York, NY 10012

[2] SRI International, 333 Ravenswood Avenue, Menlo Park, CA, 94025
E-mail address: cpeikert@alum.mit.edu
URL: http://people.csail.mit.edu/cpeikert

Abstract. We revisit the problem of generating a "hard" random lattice together with a basis of relatively short vectors. This problem has gained in importance lately due to new cryptographic schemes that use such a procedure for generating public/secret key pairs. In these applications, a shorter basis directly corresponds to milder underlying complexity assumptions and smaller key sizes.

The contributions of this work are twofold. First, using the *Hermite normal form* as an organizing principle, we simplify and generalize an approach due to Ajtai (ICALP 1999). Second, we improve the construction and its analysis in several ways, most notably by tightening the length of the output basis essentially to the optimum value.

1. Introduction

A (point) *lattice* is a discrete additive subgroup of $\mathbb{R}^m$; alternatively, it is the set of all integer linear combinations of some linearly independent *basis* vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^m$. Lattices appear to be a rich source of computational hardness, and in recent years, lattice-based *cryptographic* schemes have emerged as an intriguing alternative to more traditional ones based on, e.g., the factoring and discrete logarithm problems. Among other reasons, this is because such schemes have yet to be broken by quantum algorithms, and their security (on the average, for almost all choices of random keys) can be based solely on *worst-case* computational assumptions.

In 1996, Ajtai's seminal work [Ajt04] in this area demonstrated a family of random lattices for which finding relatively short nonzero lattice vectors is at least as hard as approximating the well-known Shortest Vector Problem (among others) in the *worst case*.

1998 ACM Subject Classification: Public key cryptosystems, Computations on discrete structures, Algorithm design and analysis.

Key words and phrases: lattices, random, short basis, average-case hardness, Hermite normal form, cryptography.

Work of the first author performed while at SRI International. This material is based upon work supported by the National Science Foundation under Grants CNS-0716786 and CNS-0749931. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

This family of "hard random lattices" has since been used as the foundation for several important cryptographic primitives, including one-way and collision-resistant hash functions, public-key encryption, digital signatures, and identity-based encryption (see, for example, [GGH96, MR07, Reg05, GPV08]).

Ajtai's initial paper also showed that a hard random lattice can be generated along with *one relatively short* lattice vector, which can be useful as a secret key in cryptographic settings (though such applications seem spare; see [MV03] for the one example of which we are aware). Shortly thereafter, Goldreich, Goldwasser and Halevi [GGH97] proposed public-key cryptographic schemes (though without security proofs) in which the secret key is a *short basis* (i.e., a basis in which all of the vectors are relatively short) of some public lattice. One method proposed in [GGH97] for generating a lattice along with a short basis is first to choose the short basis vectors, and then to transform it into a "random" public basis by a sequence of lattice-preserving transformations. Unfortunately, this method does not produce lattices from the provably hard family defined in [Ajt04]. Although improvements to the GGH lattice generator and public-key cryptosystem were later proposed by Micciancio [Mic01] (following a cryptanalysis of the original scheme by Nguyen [Ngu99]), there is still no known proof that the resulting random lattices are actually hard on the average. (We should also mention that the digital signature scheme from [GGH97] has since been shown to be insecure *regardless* of the particular method used for generating lattices [NR06].)

Following [GGH97], Ajtai demonstrated an entirely different method of generating a lattice together with a short basis [Ajt99]. This generator has the important advantage that the resulting lattice is drawn, under the appropriate distribution, from the hard family defined in [Ajt04]. Interestingly, the algorithm apparently went without application until very recently, when Gentry, Peikert and Vaikuntanathan [GPV08] constructed provably secure (under worst-case assumptions) cryptographic schemes that crucially use short bases as their secret keys; see also the subsequent works [PVW08, PV08, Pei08] for other applications. At this point we should mention that technically, the main algorithm of [Ajt99] actually produces a *full-rank set* of short lattice vectors (not necessarily a basis), which nonetheless suffices for all the applications in question.

The maximal length of the generated basis vectors directly affects the security and efficiency of the application in which it is used, both in theory and in practice. More specifically, it determines the approximation factor in the underlying worst-case lattice assumptions, as well as the concrete dimensions and key sizes needed for security against real attacks (see Section 2.1 for details). Therefore, it is very desirable to generate a set that is as short as possible. Unfortunately, the result from [Ajt99] is far from optimal — the length is bounded only by $O(m^{5/2})$, versus the optimal bound of about $\sqrt{m}$ (for commonly used parameters) — and the method appears not to have attracted much attention or improvement since its publication almost a decade ago (probably due to the lack of applications until recently).

1.1. Our Contributions

Our first contribution is to elucidate and generalize Ajtai's algorithm [Ajt99] for generating a hard random lattice along with a relatively short full-rank set of lattice vectors. We endeavor to give a high-level, modular exposition of the method and the main concerns that motivate its structure (in the process, we also correct some minor errors in the original paper). One novelty in our approach is to design and analyze the algorithm around the

concept of the *Hermite normal form* (HNF), which is a unique canonical representation for (integer) lattices. Micciancio [Mic01] has proposed using the HNF in cryptographic applications to specify a lattice in its "least revealing" representation; here we use the HNF as the central organizing tool for ensuring that the short basis corresponds to a (uniformly random) lattice from the hard family of [Ajt04].

Our second contribution is to refine the algorithm and its analysis, improving it in several ways. First and most importantly, we improve the length of its output set from $O(m^{5/2})$ to as low as $O(\sqrt{m})$, where m is the dimension of the output lattice (see Section 3 for precise statements of the new bounds). For the cryptographic schemes of, e.g. [GPV08], this immediately implies security under significantly milder worst-case assumptions: we need only that lattice problems are hard to approximate to within an $\tilde{O}(n^{3/2})$ factor, rather than $\tilde{O}(n^{7/2})$ as before. Our second main improvement is to make the generator work for an *arbitrary* integer modulus q and to output a *basis* of the resulting lattice, whereas the original algorithm of [Ajt99] works only for *odd* q and produces just a full-rank set. Using an *even* modulus q happens to be important in recent cryptosystems of Peikert [Pei08] that are based on the standard worst-case shortest vector problem. Generating a basis (versus a full-rank set) seems to be less of an advantage, but it may have unanticipated uses elsewhere.

We hasten to add that [GPV08, Section 5] mentions that Ajtai's algorithm can be improved to yield an $O(m^{1+\epsilon})$ bound on the short set, but does not provide any further details. The focus of [GPV08] is on *applications* of a short basis, independent of the particular method of its *generation*. The present work is a full exposition of an improved generation algorithm, and is meant to complement [GPV08] and other applications requiring a short basis.

1.2. Relation to Ajtai's Construction

Our construction is inspired by Ajtai's, but differs from it in most of the details. The greatest similarity is in our use of a specially crafted unimodular matrix (called $\mathbf{B}$ in this work) that has small entries, but whose inverse matrix $\mathbf{B}^{-1}$ contains geometrically increasing sequences of integers. As in [Ajt99], a crucial step in our construction involves assembling other matrices with large entries via products of short vectors and $\mathbf{B}^{-1}$.

In terms of its main differences from [Ajt99], our construction is guided from the "top down" by the abstract block structure of the short basis, the desired distribution of its Hermite normal form, and the unimodular transformation relating the two. This approach also yields various technical simplifications and corrections. In particular, it lets us completely separate the *structural constraints* on the basis from the *randomization* of the output lattice, and it facilitates a generalization to arbitrary moduli q. (In Ajtai's construction, the structure and randomization are tightly coupled, and q is assumed to be *odd* when arguing that the output set is full-rank.)

2. Preliminaries

For a positive integer k, let $[k]$ denote the set $\{1, \ldots, k\}$. We denote the set of integers modulo q by $\mathbb{Z}_q$, and identify it with the set $\{0, \ldots, q-1\}$ in the natural way. Row vectors are named by lower-case bold letters (e.g., $\mathbf{x}$) and matrices by upper-case bold letters (e.g., $\mathbf{X}$). The ith entry of a vector $\mathbf{x}$ is denoted x_i and the ith row of a matrix $\mathbf{X}$ is denoted $\mathbf{x}_i$. We identify a matrix $\mathbf{X}$ with the (ordered) set $\{\mathbf{x}_i\}$ of its row vectors, and define

$\|\mathbf{X}\| = \max_i \|\mathbf{x}_i\|$. We let $\mathbf{e}_i$ denote the ith standard basis vector, where its dimension will be clear from context. The symbol $\mathbf{I}_d$ denotes the $d \times d$ identity matrix.

2.1. Lattices

Generally defined, a *lattice* Λ is a discrete additive subgroup of $\mathbb{R}^m$ for some nonnegative integer m. In this work, every lattice will be a *full-rank integer* lattice, which is a discrete additive subgroup of $\mathbb{Z}^m$ having finite index, i.e., the quotient group $\mathbb{Z}^m/\Lambda$ is finite. The determinant of Λ, denoted $\det(\Lambda)$, is the cardinality $|\mathbb{Z}^m/\Lambda|$ of this quotient group. Geometrically, the determinant is a measure of the "sparsity" of the lattice.

A lattice $\Lambda \subseteq \mathbb{Z}^m$ can also be viewed as the set of all integer linear combinations of m linearly independent *basis* vectors $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \subset \mathbb{Z}^m$:

$$\Lambda = \mathcal{L}(\mathbf{B}) = \left\{ \mathbf{cB} = \sum_{i \in [m]} c_i \mathbf{b}_i \ : \ \mathbf{c} \in \mathbb{Z}^m \right\}.$$

A lattice has infinitely many bases (when $m \geq 2$), which are related to each other by unimodular transformations, i.e., $\mathbf{B}$ and $\mathbf{B}'$ generate the same lattice if and only if $\mathbf{B} = \mathbf{U} \cdot \mathbf{B}'$ for some unimodular matrix $\mathbf{U} \in \mathbb{Z}^{m \times m}$. The determinant of a basis matrix $\mathbf{B}$ is exactly the determinant of the lattice it generates, up to sign: $|\det(\mathbf{B})| = \det(\mathcal{L}(\mathbf{B}))$.

Every lattice $\Lambda \subseteq \mathbb{Z}^m$ has a *unique* canonical basis $\mathbf{H} = \mathrm{HNF}(\Lambda) \in \mathbb{Z}^{m \times m}$ called its *Hermite normal form* (HNF). The matrix $\mathbf{H}$ is upper triangular and has non-negative entries (i.e., $h_{i,j} \geq 0$ with equality for $i > j$), has strictly positive diagonal entries (i.e., $h_{i,i} \geq 1$), and every entry above the diagonal is strictly smaller than the diagonal entry in its column (i.e., $h_{i,j} < h_{j,j}$ for $i < j$). Note that because $\mathbf{H}$ is upper triangular, its determinant is simply the product $\prod_{i \in [m]} h_{i,i} > 0$ of its diagonal entries. For a lattice basis $\mathbf{B}$, we write $\mathrm{HNF}(\mathbf{B})$ to denote $\mathrm{HNF}(\mathcal{L}(\mathbf{B}))$. It follows that for $\mathbf{H} = \mathrm{HNF}(\mathbf{B})$, there exists a (unique) unimodular matrix $\mathbf{U}$ such that $\mathbf{B} = \mathbf{U} \cdot \mathbf{H}$. In addition, the matrices $\mathbf{U}$ and $\mathbf{H}$ can be computed in polynomial time given $\mathbf{B}$ (see [MW01] and references therein).

Hard random lattices. We will be especially concerned with a certain family of lattices in $\mathbb{Z}^m$ as first defined by Ajtai [Ajt04]. A lattice from this family is most naturally specified not by a basis, but instead by a *parity check* matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ for some positive integer n and positive integer modulus q. (We discuss the parameters m, n, and q in detail below). The associated lattice is defined as

$$\mathcal{L}^\perp(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m \ : \ \mathbf{xA} = \mathbf{0} \bmod q\} \subseteq \mathbb{Z}^m.$$

It is routine to check that $\mathcal{L}^\perp(\mathbf{A})$ contains the identity $\mathbf{0} \in \mathbb{Z}^m$ and is closed under addition, hence it is a subgroup of (and lattice in) $\mathbb{Z}^m$. Also observe that $q \cdot \mathbf{e}_i \in \mathcal{L}^\perp(\mathbf{A})$ for every $\mathbf{A}$ and every $i \in [m]$, so membership in $\mathcal{L}^\perp(\mathbf{A})$ is determined solely by a vector's entries modulo q.

We review some basic facts about this family of lattices. Let $\Lambda = \mathcal{L}^\perp(\mathbf{A})$ for some arbitrary $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$. First, we have $\det(\Lambda) \leq q^n$, by the following argument: let $\phi : (\mathbb{Z}^m/\Lambda) \to \mathbb{Z}_q^n$ be the homomorphism mapping the residue class $(\mathbf{x} + \Lambda)$ to $\mathbf{xA} \in \mathbb{Z}_q^n$. Then ϕ is injective, because if $\phi(\mathbf{x} + \Lambda) = \phi(\mathbf{x}' + \Lambda)$ for some $\mathbf{x}, \mathbf{x}' \in \mathbb{Z}^m$, we have $(\mathbf{x} - \mathbf{x}')\mathbf{A} = \mathbf{0}$ which implies $\mathbf{x} - \mathbf{x}' \in \Lambda$, i.e., $(\mathbf{x} + \Lambda) = (\mathbf{x}' + \Lambda) \in (\mathbb{Z}^m/\Lambda)$. Therefore there are at most

$|\mathbb{Z}_q^n| = q^n$ residue classes in $\mathbb{Z}^m/\Lambda$. Minkowski's first inequality states that the minimum distance of Λ (i.e., the length of a shortest nonzero lattice vector) is at most

$$\sqrt{m} \cdot \det(\Lambda)^{1/m} \leq \sqrt{m} \cdot q^{n/m}. \tag{2.1}$$

For reasons that will become clear from the statement of Proposition 2.1 below, the hardness of these lattices is most naturally parameterized by n (not m, even though m is the dimension of the lattices). Therefore, it is standard to consider the parameters $m = m(n)$ and $q = q(n)$ as functions of n. Given n and q, one of the most interesting parameter choices (which essentially minimizes the bound in (2.1)) is to let $m = c \cdot n \lg q$ for some constant $c \geq 1$. Then by (2.1), the minimum distance of $\mathcal{L}^\perp(\mathbf{A})$ for any $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ is at most

$$\sqrt{m} \cdot q^{n/m} = \sqrt{m} \cdot q^{1/(c \lg q)} = \sqrt{m} \cdot 2^{1/c} = \Theta(\sqrt{n \lg q}).$$

For a *random* $\mathbf{A}$, a volume argument reveals that with high probability, this bound is essentially tight (up to a small constant factor). Note that for larger choices of m, the minimum distance does not increase because we can just ignore the extra rows of $\mathbf{A}$. As long as m does not grow extremely large, $\sqrt{n \lg q}$ remains a good estimate for the minimum distance of $\mathcal{L}^\perp(\mathbf{A})$ for random $\mathbf{A}$.

The following proposition, proved first by Ajtai [Ajt04] (in a quantitatively weaker form) and in its current form in [MR07, GPV08], relates the average-case and worst-case complexity of certain lattice problems.

Proposition 2.1. *For any $m = m(n), \beta = \beta(n) = \text{poly}(n)$ and any $q = q(n) \geq \beta \cdot \omega(\sqrt{n \log n})$, finding a nonzero $\mathbf{x} \in \mathcal{L}^\perp(\mathbf{A})$ having length at most β for* uniformly random *$\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ (with nonnegligible probability over the choice of $\mathbf{A}$ and the randomness of the algorithm) is at least as hard as solving (with overwhelming probability) the approximate shortest vector problem GapSVP (and others) on n-dimensional lattices to within a $\gamma(n) = \beta \cdot \tilde{O}(\sqrt{n})$ factor in the* worst case.

Note that Proposition 2.1 is meaningful only when β is at least the minimum distance of a random $\mathcal{L}^\perp(\mathbf{A})$, otherwise no nonzero vector $\mathbf{x} \in \mathcal{L}^\perp(\mathbf{A})$ of length at most β is likely to exist. For $q = \text{poly}(n)$ and m as described above above, we can therefore take β to be as small as $O(\sqrt{n \lg n})$, which yields a problem that is hard on the average assuming the worst-case hardness of approximating GapSVP (and other problems) to within an $\tilde{O}(n)$ factor.

In certain cryptographic applications, however, an adversary that breaks the scheme is guaranteed only to produce lattice vectors that are much longer than the shortest vector in the lattice, so one needs to assume average-case hardness for larger values of β. For example, the secret key in the digital signature schemes of [GPV08] is a basis of $\mathcal{L}^\perp(\mathbf{A})$ having some length L, and its signatures are vectors of length $\approx L\sqrt{m}$. It is shown in [GPV08] that an adversary that is capable of forging a signature is also capable of finding a nonzero lattice vector of length $\beta \approx L\sqrt{m}$ in $\mathcal{L}^\perp(\mathbf{A})$, which by Proposition 2.1 (for our choice of m) is as hard as approximating GapSVP in the worst case to within $L \cdot \tilde{O}(n)$ factors. Therefore, a shorter secret basis immediately induces a weaker underlying hardness assumption.

Note also that Proposition 2.1 requires the modulus q to exceed β by a significant amount (otherwise the trivial vector $q \cdot \mathbf{e}_1$ would be a valid solution), and that m grows with $\lg q$. Therefore, a polynomial factor improvement in the length L of the basis also yields a constant factor improvement in the dimension m and magnitude q of entries in the parity check matrix $\mathbf{A}$ (i.e., the public key).

2.2. Probability

We denote the uniform probability distribution over a finite set G by $U(G)$. For two probability distributions D_1, D_2 (viewed as functions) over a finite set G, the statistical distance $\Delta(D_1, D_2)$ is defined to be $\frac{1}{2}\sum_{g\in G}|D_1(g) - D_2(g)|$.

Lemma 2.2 (Leftover Hash Lemma (Simplified) [HILL99])**.** *Let $\mathcal{H}$ be a family of 2-universal hash functions from a domain $\mathcal{X}$ to range $\mathcal{Y}$. and let X be a random variable over $\mathcal{X}$. Then for $h \leftarrow \mathcal{H}$ and $X \leftarrow \mathcal{X}$ chosen independently and uniformly, $(h, h(X))$ is $\frac{1}{2}\sqrt{|\mathcal{Y}| \,/\, |\mathcal{X}|}$-uniform over $\mathcal{H} \times \mathcal{Y}$.*

3. Construction

Our goal is to generate a (nearly) uniform parity check matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$, together with a basis $\mathbf{S} \in \mathbb{Z}^{m\times m}$ of $\mathcal{L}^\perp(\mathbf{A})$ whose vectors are relatively short. Our approach consists of two steps. First we investigate the structure of the Hermite normal form of $\mathcal{L}^\perp(\mathbf{A})$, for a given (random) $\mathbf{A}$. Then we describe how to generate $\mathbf{S}$ so that its HNF has the appropriate structure and distribution, so as to induce a (nearly) uniform parity check matrix $\mathbf{A}$.

We give two constructions that are, in general, incomparable. The first construction, described in Theorem 3.1 below, works for a small dimension $m = O(n \log q)$, but the resulting basis length is $\tilde{O}(m)$, which is not optimal. The second construction, described in Theorem 3.2, provides a basis of essentially *optimal* length $O(\sqrt{n \log q})$, but at the cost of a somewhat larger dimension $m = O(n \log^2 q)$. More generally, Theorem 3.2 can actually be parameterized by a base r to yield various trade-offs between the basis length and dimension m; in general, we can obtain a basis of length $\Theta(r \cdot \sqrt{n \log q})$ with a dimension $m = \Theta(n \log q \log_r q)$.

Most applications use a polynomial modulus $q = \mathrm{poly}(n)$, so the extra $\log q = O(\log n)$ factor (or $\log_r q = O(1/\delta)$ factor, when $r = n^\delta$) in the dimension m in Theorem 3.2 is of little consequence for the resulting key sizes and underlying hardness assumptions, at least asymptotically. However, certain applications (like the GapSVP-based cryptosystems of [Pei08]) in some cases rely on an exponentially large $q \approx 2^n$, in which case the extra $\log q$ factor increases the key size significantly.

Theorem 3.1. *There is a probabilistic polynomial-time algorithm that, on input a positive integer n (in unary), positive integer $q \geq 2$ (in binary), and a poly(n)-bounded positive integer $m \geq 3(1+\delta)n \lg q$ for some $\delta > 0$, outputs a pair $(\mathbf{A} \in \mathbb{Z}_q^{m\times n}, \mathbf{S} \in \mathbb{Z}^{m\times m})$ such that:*

- *$\mathbf{A}$ is $(m \cdot q^{-\delta n/2})$-uniform over $\mathbb{Z}_q^{m\times n}$,*
- *$\mathbf{S}$ is a basis of $\mathcal{L}^\perp(\mathbf{A})$, and*
- *For any $\omega(\sqrt{\log n})$ function, $\|\mathbf{S}\| \leq m \cdot \omega(\sqrt{\log n})$ with all but $n^{-\omega(1)}$ probability.*

Theorem 3.2. *There is a probabilistic polynomial-time algorithm that, on input the parameters n, q, and m as above with $m \geq 2n \lg^2 q$, outputs a pair $(\mathbf{A}, \mathbf{S})$ as above, where*

- *$\|\mathbf{S}\| \leq 5\sqrt{n \lg q}$ for every $i \in [m]$.*

The remainder of this section is devoted to proving the theorems.

3.1. Parity Check and Hermite Normal Form

As a warm-up to motivate the construction, we first consider how a given parity check matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ relates to the Hermite normal form of the lattice $\mathcal{L}^\perp(\mathbf{A})$. One may imagine that the rows $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{Z}_q^n$ of $\mathbf{A}$ are uniformly random and independent, though most of the discussion below applies to arbitrary $\mathbf{A}$.

We determine the HNF matrix $\mathbf{H} \in \mathbb{Z}^{m \times m}$ of the lattice $\Lambda = \mathcal{L}^\perp(\mathbf{A})$ inductively from the bottom up. Starting with the mth row $\mathbf{h}_m = (0, \ldots, 0, h_{m,m}) = h_{m,m} \cdot \mathbf{e}_m \in \mathbb{Z}^m$, it must be the case that

$$\mathbf{h}_m \cdot \mathbf{A} = h_{m,m} \cdot \mathbf{a}_m = \mathbf{0} \in \mathbb{Z}_q^n,$$

because every row of $\mathbf{H}$ must be in Λ. Let $k \leq q$ be the smallest positive integer solution to $k \cdot \mathbf{a}_m = \mathbf{0} \in \mathbb{Z}_q^n$. Then $k \cdot \mathbf{e}_m \in \Lambda$, so we must be able to write $k \cdot \mathbf{e}_m = \sum_{i \in [m]} z_i \mathbf{h}_i$ for some integers z_i. Now because $h_{i,i} > 0$ for every $i \in [m]$, it must therefore be the case that $z_i = 0$ for all $i < m$, which implies $h_{m,m} = k$.

Observe that when $\mathbf{a}_m$ is uniformly random, we typically have $h_{m,m} = q$, but other values of $h_{m,m}$ are also possible. For example, if q is even and every entry of $\mathbf{a}_m$ also happens to be even, then we would have $h_{m,m} \leq q/2$.

More generally, suppose that we have determined $\mathbf{h}_{i+1}, \ldots, \mathbf{h}_m$ for some $1 \leq i < m$. Then by similar reasoning, $\mathbf{h}_i \in \mathbb{Z}^m$ is given by the unique solution to the equation

$$h_{i,i} \cdot \mathbf{a}_i + \sum_{j=i+1}^{m} h_{i,j} \cdot \mathbf{a}_j = \mathbf{0} \in \mathbb{Z}_q^n$$

in which $h_{i,i} > 0$ is minimized and $0 \leq h_{i,j} < h_{j,j} \leq q$ for every $j > i$. To illustrate further, let $M_{i+1} \subseteq \mathbb{Z}_q^n$ be the subgroup of $\mathbb{Z}_q^n$ generated by (all integer linear combinations of) $\mathbf{a}_{i+1}, \ldots, \mathbf{a}_m$. Then if $\mathbf{a}_i \in M$, we have $\mathbf{a}_i = \sum_{j=i+1}^{m} z_j \mathbf{a}_j$ for some integers z_j, so $h_{i,i} = 1$, $h_{i,j} = -z_j \bmod h_{j,j}$, and $M_i = M_{i+1}$. On the other hand, if $\mathbf{a}_i \notin M$, then we have $1 < h_{i,i} \leq q$ and $M_i \supsetneq M_{i+1}$. Note that once $M_i = \mathbb{Z}_q^n$, we have $h_{i',i'} = 1$ and $h_{i',j'} = 0$ for every $i' < j' < i$.

Now suppose that $\mathbf{A}$ is uniformly random, and that $d = (1+\delta) n \lg q \leq m$ for some positive constant $\delta > 0$. Let $m' = m - d$, and break $\mathbf{A}$ into two matrices $\mathbf{A}_1 \in \mathbb{Z}_q^{m' \times n}$ and $\mathbf{A}_2 \in \mathbb{Z}_q^{d \times n}$, where $\mathbf{A}_1$ consists of the first m' rows of $\mathbf{A}$ and $\mathbf{A}_2$ consists of the remaining d. It can be shown (e.g., using the leftover hash lemma) that the rows of $\mathbf{A}_2$ generate the *entire* group $\mathbb{Z}_q^n$ with overwhelming probability over the choice of $\mathbf{A}_2$. So almost all lattices $\mathcal{L}^\perp(\mathbf{A})$ have an HNF of the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_{m'} & \mathbf{H}_1 \\ \mathbf{0} & \mathbf{H}_2 \end{bmatrix}, \tag{3.1}$$

where $\mathbf{H}_2 \in \mathbb{Z}_q^{d \times d}$ is the Hermite normal form of the lattice $\mathcal{L}^\perp(\mathbf{A}_2) \subset \mathbb{Z}^d$, which has determinant q^n. Note that there is a bijection between $\mathbb{Z}^d$ modulo $\mathbf{H}_2$ (formally, the group $\mathbb{Z}^d/(\mathbb{Z}^d \cdot \mathbf{H}_2)$) and $\mathbb{Z}_q^n$, given by $\phi(\mathbf{h}) = \mathbf{h} \cdot \mathbf{A}_2 \in \mathbb{Z}_q^n$. Note also that because $\mathbf{H} \cdot \mathbf{A} = \mathbf{0} \in \mathbb{Z}_q^{m \times n}$, we have $\mathbf{A}_1 = -\mathbf{H}_1 \cdot \mathbf{A}_2 \in \mathbb{Z}_q^{m' \times d}$. Therefore, the rows of $\mathbf{A}_1$ are uniformly random if and only if the rows of $\mathbf{H}_1$ are uniformly random modulo $\mathbf{H}_2$. Our construction (described

below) produces a basis $\mathbf{S}$ of short vectors whose HNF has the above form and a nearly identical probability distribution.

3.2. The Block Structure

To guarantee that the HNF matrix $\mathbf{H}$ of our constructed basis $\mathbf{S}$ has the desired structure and distribution, we design $\mathbf{S}$ together with the unimodular matrix $\mathbf{U}$ relating it to $\mathbf{H}$. We first set up the basic block structures of $\mathbf{S}$ and $\mathbf{U}$ according to the principal equation $\mathbf{S} = \mathbf{U} \cdot \mathbf{H}$. We then make a few simplifying choices and extract a few constraints on the blocks, and specify the blocks so as to satisfy these constraints.

Our construction first chooses $\mathbf{A}_2 \in \mathbb{Z}_q^{d \times n}$ uniformly at random and computes the HNF $\mathbf{H}_2$ of the induced lattice $\mathcal{L}^\perp(\mathbf{A}_2) \subseteq \mathbb{Z}^d$. Recall that $\mathbf{H}_2$ is nonsingular and $|\det(\mathbf{H}_2)| \leq q^n$ (note that it will usually be the case that $\mathbf{A}_2$ generates all of $\mathbb{Z}_q^n$ and $|\det(\mathbf{H}_2)| = q^n$, though we do not need this fact explicitly.) Following the form of $\mathbf{H}$ in (3.1), we obtain the following block structure on $\mathbf{S}$ and $\mathbf{U}$, where we have named the blocks of $\mathbf{S}$ for convenience.

$$\mathbf{S} = \left[\begin{array}{c:c} \mathbf{B} & \mathbf{D} \\ \hdashline \mathbf{P} & \mathbf{V} \end{array}\right] = \underbrace{\left[\begin{array}{c:c} \mathbf{B} & \mathbf{U}_{1,2} \\ \hdashline \mathbf{P} & \mathbf{U}_{2,2} \end{array}\right]}_{\mathbf{U}} \times \underbrace{\left[\begin{array}{c:c} \mathbf{I}_{m'} & \mathbf{H}_1 \\ \hdashline \mathbf{0} & \mathbf{H}_2 \end{array}\right]}_{\mathbf{H}} \tag{3.2}$$

Strictly speaking, our construction of $\mathbf{S}$ and $\mathbf{U}$ does not correspond to an $\mathbf{H}$ that is in full normal form; specifically, some entries of $\mathbf{H}_1$ might exceed their corresponding diagonal entries in $\mathbf{H}_2$. This is not a problem, because the rows of $\mathbf{H}_1$ can always be reduced modulo $\mathbf{H}_2$ via additional unimodular operations. But it is not even necessary to compute this reduced form of $\mathbf{H}_1$ in our algorithm; instead, it suffices to output $\mathbf{S}$, $\mathbf{A}_2$, and $\mathbf{A}_1 = -\mathbf{H}_1 \cdot \mathbf{A}_2 \in \mathbb{Z}_q^{m' \times n}$, and to show that the joint distribution of $(\mathbf{A}_1, \mathbf{A}_2)$ is nearly uniform.

One of the most sensitive conditions to satisfy is to make $\mathbf{U}$ unimodular. Because we only care about $\mathbf{H}_1$ modulo $\mathbf{H}_2$, the particular choices of the rightmost blocks $\mathbf{U}_{1,2}$ and $\mathbf{U}_{2,2}$ are not of much consequence. For convenience, we make $\mathbf{U}$ *block lower-triangular*, setting $\mathbf{U}_{1,2} = \mathbf{0}$ and $\mathbf{U}_{2,2} = -\mathbf{I}_d$, which implies that $\mathbf{B}$ must be unimodular. Substituting these choices, we obtain the following constraints.

$$\mathbf{H}_1 = \mathbf{B}^{-1} \cdot \mathbf{D} \tag{3.3}$$

$$\mathbf{V} + \mathbf{H}_2 = \mathbf{P} \cdot \mathbf{H}_1 = \mathbf{P} \cdot \mathbf{B}^{-1} \cdot \mathbf{D} \tag{3.4}$$

Note that the left-hand sides of the above equations have large entries, while we need all the blocks of $\mathbf{S}$ to have small entries. The $\mathbf{B}^{-1}$ term will therefore bear the sole responsibility for generating large entries. Note also the common term $\mathbf{H}_1 = \mathbf{B}^{-1} \cdot \mathbf{D}$ that appears in both equations, which causes tension between the two constraints: while we need $\mathbf{H}_1$ to be nearly uniform modulo $\mathbf{H}_2$, we also need to be able to construct $\mathbf{P}$ with small entries so that $\mathbf{P} \cdot \mathbf{H}_1$ closely approximates the matrix $\mathbf{H}_2$ that is imposed upon us.

To resolve this tension, we write $\mathbf{H}_1$ as the sum of two matrices, a random matrix $\mathbf{R}$ and a deterministic "structured" matrix $\mathbf{G}$:

$$\mathbf{H}_1 = \mathbf{B}^{-1} \cdot \mathbf{D} = \mathbf{G} + \mathbf{R}.$$

- Each row $\mathbf{r}_i$ of $\mathbf{R}$ is an independent, uniformly random vector in $\{0,1\}^d$ with random sign. We show using the leftover hash lemma that $\mathbf{R} \cdot \mathbf{A}_2$ is nearly uniform in $\mathbb{Z}_q^{m' \times n}$, hence so is $\mathbf{A}_1 = -\mathbf{H}_1 \cdot \mathbf{A}_2$.
- The matrix $\mathbf{G}$ is designed so that small integer combinations of its rows may be assembled to produce (a matrix close to) $\mathbf{H}_2$; more specifically,

  $$\mathbf{P} \cdot \mathbf{G} = \mathbf{H}_2' = \mathbf{H}_2 - \mathbf{I}_d$$

  for some $\mathbf{P}$ having small entries (we subtract $\mathbf{I}_d$ from $\mathbf{H}_2$ simply for convenience, to put the diagonals of $\mathbf{H}_2'$ in the range $\{0, \ldots, q-1\}$). Furthermore, $\mathbf{G}$ and $\mathbf{B}$ are designed together to make $\mathbf{B} \cdot \mathbf{G}$ have small entries, so that

  $$\mathbf{D} = \mathbf{B} \cdot \mathbf{H}_1 = \mathbf{B} \cdot \mathbf{G} + \mathbf{B} \cdot \mathbf{R}$$

  has small entries as well.

We then let $\mathbf{V} = \mathbf{P} \cdot \mathbf{R} - \mathbf{I}_d$; observe that $\mathbf{V}$ has small entries because $\mathbf{P}$, $\mathbf{R}$, and $\mathbf{I}_d$ do, and that (3.4) is satisfied because

$$\mathbf{P} \cdot \mathbf{H}_1 = \mathbf{P} \cdot (\mathbf{G} + \mathbf{R}) = \mathbf{H}_2 + \mathbf{V}.$$

3.3. Building the Blocks

Here we list the principal constraints on the as-yet undefined matrices $\mathbf{B}$, $\mathbf{P}$, and $\mathbf{G}$ from the above discussion, and show how to satisfy those constraints.

(1) Matrix $\mathbf{B}$ must be unimodular and have small entries.
(2) The product $\mathbf{W} = \mathbf{B} \cdot \mathbf{G}$ must have small entries.
(3) We must satisfy $\mathbf{P} \cdot \mathbf{G} = \mathbf{H}_2' = \mathbf{H}_2 - \mathbf{I}_d$ for some $\mathbf{P}$ with small entries.

Below we give two constructions, corresponding to Theorems 3.1 and 3.2, respectively. In both constructions, we assemble $\mathbf{B}$ from copies of a certain component matrix $\mathbf{T}_k \in \mathbb{Z}^{k \times k}$, which is defined to be the $k \times k$ lower-triangular matrix with 1s along the diagonal, -2s directly below the diagonal, and 0s elsewhere, i.e., $t_{i,i} = 1$ for $i \in [k]$ and $t_{i+1,i} = -2$ for $i \in [k-1]$. It may be verified that $\mathbf{T}_k$ is lower triangular and unimodular. Moreover, its inverse $\mathbf{T}_k^{-1}$ has a very useful form: its (i,j)th entry is 2^{i-j} for every $i \geq j$, and zero elsewhere.

3.3.1. *Construction for Theorem 3.1.* Define $m' = m - d \geq 2d$. The basic idea is to construct $\mathbf{G} = \mathbf{B}^{-1} \cdot \mathbf{W} \in \mathbb{Z}^{m' \times d}$ so that it contains enough power-of-2 multiples of each of the standard basis vectors in $\mathbb{Z}^d$; this is done by assembling $\mathbf{B}$ from copies of $\mathbf{T}_k$ and letting $\mathbf{W}$ have small entries, thus satisfying constraint 2. Then any vector in $\mathbb{Z}^d$ with bounded entries (specifically, every row of $\mathbf{H}_2'$) can be expressed as a binary combination of the rows of $\mathbf{G}$, thus satisfying constraint 3.

We now proceed in more detail. Recall that we are given $\mathbf{H}_2 \in \mathbb{Z}^{d \times d}$; say its diagonal entries (from top to bottom) are $r_1, \ldots, r_d$, and recall that their product is (at most) q^n. Let $\ell_j = \lceil \lg r_j \rceil \leq 1 + \lg r_j$, and define the partial sums $s_0 = 0$, $s_j = s_{j-1} + \ell_j$ for $j \in [d]$, and define the total sum $s = s_d \leq d + n \lg q \leq m'$.

Define $\mathbf{B} \in \mathbb{Z}^{m' \times m'}$ to be the block diagonal matrix

$$\mathbf{B} = \mathrm{diag}(\mathbf{T}_{\ell_1}, \ldots, \mathbf{T}_{\ell_d}, \mathbf{I}_{m'-s}),$$

i.e., the direct sum of $\mathbf{T}_{\ell_j}$ for $j \in [d]$, plus an identity matrix of the appropriate remaining dimension. Observe that $\mathbf{B}$ is lower triangular and unimodular, and that $\mathbf{B}^{-1} = \mathrm{diag}(\mathbf{T}_{\ell_1}^{-1}, \ldots, \mathbf{T}_{\ell_d}^{-1}, \mathbf{I}_{m'-s})$.

Now define $\mathbf{W}$ so that $\mathbf{w}_{s_{j-1}+1} = \mathbf{e}_j \in \mathbb{Z}^d$ for each $j \in [d]$, and $\mathbf{w}_i = \mathbf{0}$ elsewhere. Recalling that $\mathbf{G} = \mathbf{B}^{-1} \cdot \mathbf{W}$, one can then check that for each $j \in [d]$ and each $k \in [\ell_j]$, we have

$$\mathbf{g}_{s_{j-1}+k} = 2^{k-1} \cdot \mathbf{e}_j \in \mathbb{Z}^d$$

(and $\mathbf{g}_i = \mathbf{0}$ for $s < i \leq m'$).

Because $\mathbf{G}$ has such a useful form, satisfying constraint 3 (i.e., making $\mathbf{P} \cdot \mathbf{G} = \mathbf{H}_2 - \mathbf{I}_d$) is straightforward. For each $j \in [d]$, every entry of the jth column of $\mathbf{H}_2'$ is in $\{0, \ldots, r_j - 1\}$, by construction of $\mathbf{H}_2$. Therefore, each row of $\mathbf{H}_2'$ can be represented as a binary combination of rows $\mathbf{g}_1, \ldots, \mathbf{g}_s$ of $\mathbf{G}$. These binary combinations are specified in the natural way via the d rows of $\mathbf{P}$, and we have satisfied constraint 3 where each entry of $\mathbf{P}$ has magnitude at most 1.

3.3.2. *Construction for Theorem 3.2.* Define $m' = m - d \geq d \cdot \lceil \lg q \rceil$. The basic idea is to construct $\mathbf{G} = \mathbf{B}^{-1} \cdot \mathbf{W} \in \mathbb{Z}^{m' \times d}$ so that $\mathbf{G}$ *itself* contains the rows of $\mathbf{H}_2'$, which can then be trivially selected by very short rows $\mathbf{p}_i$ having length 1 (rather than almost $\sqrt{m}$ as above). To do this, we let $\mathbf{B}$ be made up of copies of $\mathbf{T}_k$ much like above, and let $\mathbf{W}$ encode the binary representation of each row of $\mathbf{H}_2'$. Note that $\mathbf{H}_2'$ has d rows with entries that can be as large as $q - 1$, so we can represent it in binary using $d \cdot \lceil \lg q \rceil \leq m'$ rows. (More generally, using the base-r analog of $\mathbf{T}_k$ instead of base 2, we can represent $\mathbf{H}_2'$ using $d \cdot \log_r q$ rows, at the expense of using vectors $\mathbf{b}_i$ having length $O(r)$.)

Define $\ell = \lceil \lg(q-1) \rceil$ and define $\mathbf{B} \in \mathbb{Z}^{m' \times m'}$ be the block diagonal matrix

$$\mathbf{B} = \mathrm{diag}(\mathbf{T}_\ell, \ldots, \mathbf{T}_\ell, \mathbf{I}_{m'-d\cdot\ell})$$

(where the above expression includes d copies of $\mathbf{T}_\ell$). Observe that $\mathbf{B}$ is lower triangular and unimodular, and that $\mathbf{B}^{-1} = \mathrm{diag}(\mathbf{T}_\ell^{-1}, \ldots, \mathbf{T}_\ell^{-1}, \mathbf{I}_{m'-d\cdot\ell})$.

We now define $\mathbf{W}$. Let $\mathbf{h}_j' \in \mathbb{Z}^d$ denote the jth row of $\mathbf{H}_2'$, and observe that every entry of $\mathbf{h}_j'$ is nonnegative and at most $q-1$, so it can be written in binary using ℓ bits. Therefore $\mathbf{h}_j'$ can be seen as the ℓth row of $\mathbf{T}_\ell^{-1} \cdot \mathbf{W}_j$ for a binary matrix $\mathbf{W}_j \in \{0,1\}^{\ell \times d}$, where the rows of $\mathbf{W}_j$ consist of the coordinate-wise bits of $\mathbf{h}_j'$ from most significant down to least significant. Finally, let $\mathbf{W} \in \mathbb{Z}^{m' \times d}$ be the vertical block matrix consisting of $\mathbf{W}_1$ through $\mathbf{W}_d$, followed by the zero matrix of dimension $(m' - d \cdot \ell) \times d$. Then for $\mathbf{G} = \mathbf{B}^{-1} \cdot \mathbf{W}$, it is apparent from the above discussion that row $\mathbf{g}_{j\cdot\ell} = \mathbf{h}_j'$ for each $j \in [d]$. The corresponding rows of $\mathbf{P}$ are $\mathbf{p}_j = \mathbf{e}_{j\cdot\ell} \in \mathbb{Z}^{m'}$ for $j \in [d]$.

3.4. Analysis

We now prove that the above constructions satisfy the claims in Theorems 3.1 and 3.2, respectively. We have already shown by construction that $\mathbf{S}$ is a basis of $\mathcal{L}^\perp(\mathbf{A})$. It remains to show that the distribution of $\mathbf{A}$ is statistically close to uniform over $\mathbb{Z}_q^{m \times n}$, and that the rows of $\mathbf{S}$ are all relatively short (in both constructions).

3.4.1. *Distribution of* $\mathbf{A}$. Recall that in both constructions, $\mathbf{A}$ is of the form

$$(\mathbf{A}_1 = -\mathbf{H}_1 \cdot \mathbf{A}_2 \ , \ \mathbf{A}_2) \ = \ (-(\mathbf{G} + \mathbf{R}) \cdot \mathbf{A}_2 \ , \ \mathbf{A}_2) \ \in \ \mathbb{Z}_q^{m \times n},$$

where $\mathbf{A}_2 \in \mathbb{Z}_q^{d \times n}$ is uniform, $\mathbf{G}$ is deterministic, and each row of $\mathbf{R}$ is independent and uniform from $\{0,1\}^d$ (with random sign).

We claim that $\{h_{\mathbf{A}_2} : h_{\mathbf{A}_2}(\mathbf{r}) = \mathbf{r}\mathbf{A}_2\}$ is a family of 2-universal hash functions from domain $\{0,1\}^d$ to range $\mathbb{Z}_q^n$. First, note that $\mathbf{r}\mathbf{A}_2 = \mathbf{r}'\mathbf{A}_2$ if and only if $(\mathbf{r} - \mathbf{r}')\mathbf{A}_2 = \mathbf{0}$, and $\mathbf{0} \neq \mathbf{r} - \mathbf{r}' \in \{0, \pm 1\}$ for any distinct $\mathbf{r}, \mathbf{r}' \in \{0,1\}$. Fix such $\mathbf{r}, \mathbf{r}'$, and suppose that they differ in their ith entry. Finally, observe that

$$\Pr_{\mathbf{A}_2}[(\mathbf{r} - \mathbf{r}')\mathbf{A}_2 = \mathbf{0}] = q^{-n} = 1/\left|\mathbb{Z}_q^n\right|,$$

by averaging over any fixed choice of all but the ith row of $\mathbf{A}_2$.

Now because $d = (1+\delta)n \lg q$ for some constant $\delta > 0$, Lemma 2.2 (the leftover hash lemma) and the triangle inequality imply that $(\mathbf{R} \cdot \mathbf{A}_2, \mathbf{A}_2)$ is $(m \cdot q^{-\delta n/2})$-uniform over $\mathbb{Z}_q^{m \times n}$, as desired.

3.4.2. *Length of* $\mathbf{S}$. We need to analyze the lengths of the rows of $\mathbf{B}$, $\mathbf{P}$, $\mathbf{D}$, and $\mathbf{V}$, where

$$\begin{aligned} \mathbf{D} &= \mathbf{B}\mathbf{G} + \mathbf{B}\mathbf{R} \\ \mathbf{V} &= \mathbf{P}\mathbf{R} - \mathbf{I}_d \end{aligned}$$

- In both constructions, $\mathbf{B}\mathbf{G} = \mathbf{W}$ is a binary matrix (or in the base-r generalization of Theorem 3.2, an r-ary matrix). Thus $\|\mathbf{B}\mathbf{G}\| \leq \sqrt{d}$ (more generally, $(r-1)\sqrt{d}$).
- We have $\|\mathbf{R}\| \leq \sqrt{d}$ by construction, and the ℓ_1 norm (i.e., the sum of the absolute values of each entry) of each $\mathbf{b}_i$ is at most 3 (more generally, at most $r+1$). So by the triangle inequality, we have $\|\mathbf{B}\mathbf{R}\| \leq 3\sqrt{d}$ (more generally, $(r+1)\sqrt{d}$).
- Note that $\|\mathbf{V}\| \leq \|\mathbf{P}\mathbf{R}\| + 1$ by the triangle inequality.

It remains to analyze $\|\mathbf{P}\mathbf{R}\|$ for the two constructions. In the construction for Theorem 3.2, each $\mathbf{p}_i$ has just a single 1 entry (and 0s elsewhere), so $\|\mathbf{P}\mathbf{R}\| \leq \sqrt{d}$. Putting all the blocks of $\mathbf{S}$ together, we conclude that in the construction for Theorem 3.2, $\|\mathbf{S}\| \leq (2r+1)\sqrt{d}$, as desired.

We now analyze the construction for Theorem 3.1. Let s be the random variable corresponding to any entry of $\mathbf{P}\mathbf{R}$. Because $\mathbf{P}$ is a fixed binary matrix, s is the sum of at most m independent random variables $r_{i,j}$ that individually have expectation 0 (because the sign of each $\mathbf{r}_i$ is random) and magnitude at most 1. By the Hoeffding bound, we have $|s| \leq t \cdot \sqrt{m}$ except with probability at most $\exp(-\Omega(t^2))$. Setting $t = \omega(\sqrt{\log n})$ and taking a union bound over all $\mathrm{poly}(n)$ entries of $\mathbf{P}\mathbf{R}$, we conclude that $\|\mathbf{P}\mathbf{R}\| \leq t \cdot \sqrt{m \cdot d} \leq t \cdot m$, except with probability $n^{-\omega(1)}$, as desired.

References

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999.

[Ajt04] Miklós Ajtai. Generating hard instances of lattice problems. *Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.

[GGH96] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(42), 1996.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *CRYPTO*, pages 112–131, 1997.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[Mic01] Daniele Micciancio. Improving lattice based cryptosystems using the Hermite normal form. In *CaLC*, pages 126–145, 2001.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[MV03] Daniele Micciancio and Salil P. Vadhan. Statistical zero-knowledge proofs with efficient provers: Lattice problems and more. In *CRYPTO*, pages 282–298, 2003.

[MW01] Daniele Micciancio and Bogdan Warinschi. A linear space algorithm for computing the Hermite normal form. In *ISSAC*, pages 231–236, 2001.

[Ngu99] Phong Q. Nguyen. Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto '97. In *CRYPTO*, pages 288–304, 1999.

[NR06] Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In *EUROCRYPT*, pages 271–288, 2006.

[Pei08] Chris Peikert. Public key cryptosystems from the worst-case shortest vector problem. In submission, 2008.

[PV08] Chris Peikert and Vinod Vaikuntanathan. Noninteractive statistical zero-knowledge proofs for lattice problems. In *CRYPTO*, pages 536–553, 2008.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 87–98
www.stacs-conf.org

QUANTUM QUERY COMPLEXITY OF MULTILINEAR IDENTITY TESTING

V. ARVIND [1] AND PARTHA MUKHOPADHYAY [1]

[1] The Institute of Mathematical Sciences
CIT Campus, Taramani
Chennai, India 600113
E-mail address: arvind@imsc.res.in
E-mail address: partham@imsc.res.in
URL: http://www.imsc.res.in

ABSTRACT. Motivated by the quantum algorithm for testing commutativity of black-box groups (Magniez and Nayak, 2007), we study the following problem: Given a black-box finite ring by an additive generating set and a multilinear polynomial over that ring, also accessed as a black-box function (we allow the indeterminates of the polynomial to be commuting or noncommuting), we study the problem of testing if the polynomial is an *identity* for the given ring. We give a quantum algorithm with query complexity sub-linear in the number of generators for the ring, when the number of indeterminates of the input polynomial is small (ideally a constant). Towards a lower bound, we also show a reduction from a version of the collision problem (which is well studied in quantum computation) to a variant of this problem.

1. Introduction

For any finite ring $(R, +, \cdot)$ the ring $R[x_1, x_2, \cdots, x_m]$ is the ring of polynomials in commuting variables $x_1, x_2, \cdots, x_m$ and coefficients in R. The ring $R\{x_1, x_2, \cdots, x_m\}$ is the ring of polynomials where the indeterminates x_i are *noncommuting*. By noncommuting variables, we mean $x_i x_j - x_j x_i \neq 0$ for $i \neq j$.

For the algorithmic problem we study in this paper, we assume that the elements of the ring $(R, +, \cdot)$ are uniformly encoded by binary strings of length n and $R = \langle r_1, r_2, \cdots, r_k \rangle$ is given by an additive generating set $\{r_1, r_2, \cdots, r_k\}$. That is,

$$R = \{\sum_i \alpha_i r_i \mid \alpha_i \in \mathbb{Z}\}.$$

Also, the ring operations of R are performed by black-box oracles for addition and multiplication that take as input two strings encoding ring elements and output their sum or product (as the case may be). Additionally, we assume that the zero element of R is encoded by a fixed string. The black-box model for finite rings was introduced in [ADM06]. We now define the problem which we study in this paper.

1998 ACM Subject Classification: F.2.1 Computation on Polynomials.
Key words and phrases: Quantum Algorithm, Identity Testing, Query Complexity, Multilinear Polynomials.

The Multilinear Identity Testing Problem (MIT**):** The input to the problem is a black-box ring $R = \langle r_1, \cdots, r_k \rangle$ given by an additive generating set, and a multilinear polynomial $f(x_1, \cdots, x_m)$ (in the ring $R[x_1, \cdots, x_m]$ or the ring $R\{x_1, \cdots, x_m\}$) that is also given by a black-box access. The problem is to test if f is an *identity* for the ring R. More precisely, the problem is to test if $f(a_1, a_2, \cdots, a_m) = 0$ for all $a_i \in R$.

A natural example of an instance of this problem is the bivariate polynomial $f(x_1, x_2) = x_1x_2 - x_2x_1$ over the ring $R\{x_1, x_2\}$. This is an identity for R precisely when R is a commutative ring. Clearly, it suffices to check if the generators commute with each other, which gives a naive algorithm that makes $O(k^2)$ queries to the ring oracles.

Given a polynomial $f(x_1, \cdots, x_m)$ and a black-box ring R by generators, we briefly discuss some facts about the complexity of checking if $f = 0$ is an identity for R. The problem can be NP-hard when the number of indeterminates m is unbounded, even when R is a fixed ring. To see this, notice that a 3-CNF formula $F(x_1, \cdots, x_n)$ can be expressed as a $O(n)$ degree multilinear polynomial $f(x_1, x_2, \cdots, x_n)$ over $\mathbb{F}_2$, by writing F in terms of addition and multiplication over $\mathbb{F}_2$. It follows that $f = 0$ is an identity for $\mathbb{F}_2$ if and only if F is an unsatisfiable formula. However in this paper we focus only on the upper and lower bounds on the *query complexity* of the problem.

In our query model, each ring operation, which is performed by a query to one of the ring oracles, is of unit cost. Furthermore, we consider each evaluation of $f(a_1, \cdots, a_m)$ to be of unit cost for a given input $(a_1, \cdots, a_m) \in R^m$. This model is reasonable because we consider m as a parameter that is much smaller than k.

The starting point of our study is a result of Magniez and Nayak in [MN07], where the authors study the quantum query complexity of group commutativity testing: Let G be a finite black-box group given by a generating set $g_1, g_2, \cdots, g_k$ and the group operation is performed by a group oracle. The algorithmic task is to check if G is commutative. For this problem the authors in [MN07] give a quantum algorithm with query complexity $O(k^{2/3} \log\ k)$ and time complexity $O(k^{2/3} \log^2\ k)$. Furthermore, a $\Omega(k^{2/3})$ lower bound for the quantum query complexity is also shown. The main technical tool for their upper bound result was a method of quantization of random walks first shown by Szegedy [Sze04]. More recently, Magniez et al in [MNRS07] discovered a simpler and improved description of Szegedy's method.

Our starting point is the observation that Magniez-Nayak result [MN07] for group commutativity can also be easily seen as a commutativity test for arbitrary finite black-box *rings* with similar query complexity. Furthermore, as mentioned earlier, notice that the commutativity testing for a finite ring coincides with testing if the bivariate polynomial $f(x_1, x_2) = x_1x_2 - x_2x_1$ is an identity for the ring. Since $f(x_1, x_2)$ is a multilinear polynomial, a natural question is, whether this approach would extend to testing if any multilinear polynomial is an identity for a given ring. Motivated by this connection, we study the problem of testing multilinear identities for any finite black-box ring.

The upper bound result in [MN07] is based on a group-theoretic lemma of Pak [Pak00]. Our (query complexity) upper bound result takes an analogous approach. The main technical contribution here is a suitable generalization of Pak's lemma to a multilinear polynomial setting. The multilinearity condition is crucially required. The rest of the proof is a suitable adaptation of the Magniez-Nayak result.

For the lower bound result, we show a reduction to a somewhat more general version of MIT from a problem that is closely related to the m-COLLISION problem studied in quantum computation. The m-COLLISION problem is the following. Given a function $f : \{1, 2, \cdots, k\} \rightarrow \{1, 2, \cdots, k\}$ as an oracle and a positive integer m, the task is to determine if there is some element in the range of f with exactly m pre-images.

We define the m-SPLIT COLLISION problem that is closely related to m-COLLISION problem. Here the domain $\{1, 2, \cdots, k\}$ is partitioned into m equal-sized intervals (assume k is a multiple of m) and the problem is to determine if there is some element in the range of f with exactly one pre-image in each of the m intervals. We show a reduction from m-SPLIT COLLISION to a general version of MIT. There is an easy randomized reduction from m-COLLISION problem to m-SPLIT COLLISION problem. The best known quantum query complexity lower bound for m-COLLISION problem is $\Omega(k^{\frac{2}{3}})$ [AS04] and thus we get the same lower bound for the general version of MIT that we study. Improving, the current lower bound for m-COLLISION is an important open problem in quantum computation since last few years. [1]

Our reduction for lower bound is conceptually different from the lower bound proof in [MN07]. It uses ideas from automata theory to construct a suitable black-box ring. We recently used similar ideas in the design of a deterministic polynomial-time algorithm for identity testing of noncommutative circuits computing small degree sparse polynomials [AMS08].

2. Black-box Rings and the Quantum Query model

We briefly explain the standard quantum query model. We modify the definition of black-box ring operations by making them unitary transformations that can be used in quantum algorithms. For a black-box ring R, we have two oracles O_R^a and O_R^m for addition and multiplication respectively. For any two ring elements r, s, and a binary string $t \in \{0,1\}^n$ we have $O_R^a|r\rangle|s\rangle = |r\rangle|r+s\rangle$ and $O_R^m|r\rangle|s\rangle|t\rangle = |r\rangle|s\rangle|rs \oplus t\rangle$, where the elements of R are encoded as strings in $\{0,1\}^n$. Notice that O_R^a is a reversible function by virtue of $(R, +)$ being an additive group. On the other hand, $(R, \cdot)$ does not have a group structure. Thus we have made O_R^m reversible by defining it as a 3-place function $O_R^m : \{0,1\}^{3n} \to \{0,1\}^{3n}$. When r or s do not encode ring elements these oracles can compute any arbitrary string.

The query model in quantum computation is a natural extension of classical query model. The basic difference is that a classical algorithm queries deterministically or randomly selected basis states, whereas a quantum algorithm can query a quantum state which is a suitably prepared superposition of basis states. Our query model closely follows the query model of Magniez-Nayak [MN07, Section 2.2]. For black-box ring operations the query operators are simply O_R^a and O_R^m (as defined above). For an arbitrary oracle function $F : X \to Y$, the corresponding unitary operator is $O_F : |g\rangle|h\rangle \to |g\rangle|h \oplus F(g)\rangle$. In the query complexity model, we charge unit cost for a single query to the oracle and all other computations are free. We will assume that the input black-box polynomial $f : R^m \to R$ is given by such an unitary operator U_f.

All the quantum registers used during the computation can be initialised to $|0\rangle$. Then a k-query algorithm for a black-box ring is a sequence of $k+1$ unitary operators and k ring oracle operators: $U_0, Q_1, U_1, \cdots, U_{k-1}, Q_k, U_k$ where $Q_i \in \{O_R^a, O_R^m, O_F\}$ are the oracle queries and U_i's are unitary operators. The final step of the algorithm is to measure designated qubits and decide according to the measurement output.

3. Quantum Algorithm for Multilinear Identity Testing

In this section we describe our quantum algorithm for multilinear identity testing (MIT). Our algorithm is motivated by (and based on) the group commutativity testing algorithm of Magniez and Nayak [MN07]. We briefly explain the algorithm of Magniez-Nayak. Their problem is the

[1] Ambainis in [Amb07] show a quantum query complexity upper bound of $O(k^{m/m+1})$ for m-COLLISION problem.

following: given a black-box group G by a set of generators $g_1, g_2, \cdots, g_k$, the task is to find nontrivial upper bound on the quantum query complexity to determine whether G is commutative. The group operators (corresponding to the oracle) are O_G and $O_{G^{-1}}$.

Note that for this problem, there is a trivial classical algorithm (so as quantum) of query complexity $O(k^2)$. In an interesting paper, Pak showed a classical randomized algorithm of query complexity $O(k)$ for the same problem [Pak00]. Pak's algorithm is based on the following observation ([Pak00, Lemma 1.3]): Consider a subproduct $h = g_1^{e_1} g_2^{e_2} \cdots g_k^{e_k}$ where e_i' s are picked uniformly at random from $\{0, 1\}$. Then for any proper subgroup H of G, $\text{Prob}[h \notin H] \geq 1/2$.

One important step of the algorithm in [MN07] is a generalization of Pak's lemma. Let $\mathcal{V}_\ell$ be the set of all distinct element ℓ tuples of elements from $\{1, 2, \cdots, k\}$. For $u = (u_1, \cdots, u_\ell)$, define $g_u = g_{u_1} \cdot g_{u_2} \cdots g_{u_\ell}$. Let $p = \frac{\ell(\ell-1)+(k-\ell)(k-\ell-1)}{k(k-1)}$.

Lemma 3.1. [MN07] *For any proper subgroup K of G,* $\text{Prob}_{u \in \mathcal{V}_\ell}[g_u \notin K] \geq \frac{1-p}{2}$.

As a simple corollary of this lemma, Magniez and Nayak show in [MN07] that, if G is non abelian then for randomly picked u and v from $\mathcal{V}_\ell$ the elements g_u and g_v will not commute with probability at least $\frac{(1-p)^2}{4}$. Thus, for non abelian G there will be at least $\frac{(1-p)^2}{4}$ fraction of noncommuting pairs (u, v). Call such pairs as *marked pairs*. Next, their idea is to do a random walk in the space of all pairs and to decide whether there exists a marked pair. They achieved this by defining a random walk and quantizing it using [Sze04]. We briefly recall the setting from [MN07, Section 2.3], and the main theorem from [Sze04], which is the central to the analysis of Magniez-Nayak result.

3.0.1. *Quantum Walks.* Let P be an irreducible and aperiodic Markov chain on a graph $G = (V, E)$ with n vertices. A walk following such a Markov chain is always ergodic and has unique stationary distribution. Let $P(u, v)$ denote the transition probability from $u \to v$, and M be a set of marked nodes of V. The goal is to make a walk on the vertices of G following the transition matrix P and decide whether M is *nonempty*. Assume that every node $v \in V$ is associated with a database $D(v)$ from which we can determine whether $v \in M$. This search procedure is modelled by a quantum walk. To analyze the performance of the search procedure, we need to consider the cost of the following operations:

Set up Cost (S): The cost to set up $D(v)$ for $v \in V$.

Update Cost (U): The cost to update $D(v)$, i.e. to update from $D(v)$ to $D(v')$, where the move $v \to v'$ is according to the transition matrix P.

Checking Cost (C): To check whether $v \in M$ using $D(v)$.

The costs are specific to the application for e.g. it can be query complexity or time complexity. The problem that we consider or the group commutativity problem of Magniez-Nayak, concern about query complexity. The following theorem due to Szegedy gives a precise analysis of the total cost involved in the quantum walk.

Theorem 3.2. [Sze04] *Let P be the transition matrix of an ergodic, symmetric Markov Chain on a graph $G = (V, E)$ and δ be the spectral gap of P. Also, let M be the set of all marked vertices in V and $|M|/|V| \geq \epsilon > 0$, whenever M is nonempty. Then there is a quantum algorithm which determines whether M is nonempty with constant success probability and cost $S+O((U+C)/\sqrt{\delta\epsilon})$. S is the set up cost of the quantum process, U is the update cost for one step of the walk and C is the checking cost.*

Later, Magniez-Nayak-Ronald-Santha [MNRS07] improve the total cost of the quantum walk. We state their main result.

Theorem 3.3. [MNRS07] *Let P be the transition matrix of a reversible, ergodic Markov Chain on a graph $G = (V, E)$ and δ be the spectral gap of P. Also let M be the set of all marked vertices in V and $|M|/|V| \geq \epsilon > 0$, whenever M is nonempty. Then there is a quantum algorithm which determines whether M is nonempty and in that case finds an element of M, with constant success probability and cost of order $S + \frac{1}{\sqrt{\epsilon}}(\frac{1}{\sqrt{\delta}}U + C)$. S is the set up cost of the quantum process, U is the update cost for one step of the walk and C is the checking cost.*

The analysis of Magniez-Nayak [MN07] is based on Theorem 3.2. For our problem also, we follow similar approach.

3.1. Query Complexity Upper Bound

Now we describe our quantum algorithm for MIT. Our main technical contribution is a suitable generalization of Pak's lemma. For any $i \in [m]$, consider the set $R_i \subseteq R$ defined as follows:

$$R_i = \{u \in R \mid \forall (b_1, \cdots, b_{i-1}, b_{i+1}, \cdots, b_m) \in R^{m-1}, f(b_1, \cdots, b_{i-1}, u, b_{i+1}, \cdots, b_m) = 0\}$$

Clearly, if f is not a zero function from $R^m \to R$, then $|R_i| < |R|$. In the following lemma, we prove that if f is not a zero function then $|R_i| \leq |R|/2$.

Lemma 3.4. *Let R be any finite ring and $f(x_1, x_2, \cdots, x_m)$ be a multilinear polynomial over R such that $f = 0$ is not an identity for R. For $i \in [m]$ define*

$$R_i = \{u \in R \mid \forall (b_1, \cdots, b_{i-1}, b_{i+1}, \cdots, b_m) \in R^{m-1}, f(b_1, \cdots, b_{i-1}, u, b_{i+1}, \cdots, b_m) = 0\}.$$

Then R_i is an additive coset of a proper additive subgroup of R and hence $|R_i| \leq |R|/2$.

Proof. Write $f = A(x_1, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_m) + B(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_m)$ where A is the sum of all the monomials of f containing x_i and B is the sum of the rest of the monomials. Let v_1, v_2 be any two distinct elements in R_i. Then for any fixed $\bar{y} = (y_1, \cdots, y_{i-1}, y_{i+1}, \cdots, y_m) \in R^{m-1}$, consider the evaluation of A and B over the points $(y_1, \cdots, y_{i-1}, v_1, y_{i+1}, \cdots, y_m)$ and $(y_1, \cdots, y_{i-1}, v_2, y_{i+1}, \cdots, y_m)$ respectively. For convenience, we abuse the notation and write,

$$A(v_1, \bar{y}) + B(\bar{y}) = A(v_2, \bar{y}) + B(\bar{y}) = 0,$$

where $\bar{y}$ is an assignment to $x_1, x_2, \cdots, x_{i-1}, x_{i+1}, \cdots, x_k$ and v_1, v_2 are the assignments to x_i respectively. Note that, as f is a multilinear polynomial, the above relation in turns implies that $A(v_1 - v_2, \bar{y}) = 0$.

Consider the set $\hat{R}_i$, defined as follows: Fix any $u^{(i)} \in R_i$,

$$\hat{R}_i = \{w - u^{(i)} \mid w \in R_i\}.$$

We claim that $\hat{R}_i$ is an (additive) subgroup of R. We only need to show that $\hat{R}_i$ is closed under the addition (of R). Consider $(w_1 - u^{(i)}), (w_2 - u^{(i)}) \in \hat{R}_i$. Then $(w_1 - u^{(i)}) + (w_2 - u^{(i)}) = (w_1 + w_2 - u^{(i)}) - u^{(i)}$. It is now enough to show that for any $\bar{y} \in R^{m-1}$, $f(w_1 + w_2 - u^{(i)}, \bar{y}) = 0$ (note that $w_1 + w_2 + u^{(i)}$ is an assignment to x_i). Again using the fact that f is multilinear, we can easily see the following:

$$f(w_1 + w_2 - u^{(i)}, \bar{y}) = A(w_1, \bar{y}) + A(w_2, \bar{y}) - A(u^{(i)}, \bar{y}) + B(\bar{y})$$

and,

$$A(w_1, \bar{y}) + A(w_2, \bar{y}) - A(u^{(i)}, \bar{y}) + B(\bar{y}) = A(w_2, \bar{y}) - A(u^{(i)}, \bar{y}) = 0.$$

Note that the last equality follows because x_2 and u are in R_i. Hence we have proved that $\hat{R}_i$ is a subgroup of R. So $R_i = \hat{R}_i + u^{(i)}$ i.e. R_i is a coset of $\hat{R}_i$ inside R. Also $|R_i| < |R|$ (f is not identically zero over R). Thus, finally we get $|R_i| = |\hat{R}_i| \leq |R|/2$. ∎

Our quantum algorithm is based on the algorithm of [MN07]. In the rest of the paper we denote by S_ℓ the set of all ℓ size *subsets* of $\{1, 2, \cdots, k\}$. We follow a quantization of a random walk on $S_\ell \times \cdots \times S_\ell = S_\ell^m$. For $u = \{u_1, u_2, \cdots, u_\ell\}$, define $r_u = r_{u_1} + \cdots + r_{u_\ell}$. Now, we suitably adapt Lemma 1 of [MN07] in our context. [2]

Let R be a finite ring given by a additive generating set $S = \{r_1, \cdots, r_k\}$. W.l.o.g. assume that r_1 is the zero element of R. Let $\hat{R}$ be a proper additive subgroup of $(R, +)$. Let j be the least integer in $[k]$ such that $r_j \notin \hat{R}$. Since $\hat{R}$ is a proper subgroup of R, such a j always exists.

Lemma 3.5. *Let $\hat{R} < R$ be a proper additive subgroup of R and T be an additive coset of $\hat{R}$ in R. Then* $\text{Prob}_{u \in S_\ell}[r_u \notin T] \geq \frac{1-p}{2}$, *where* $p = \frac{\ell(\ell-1)+(k-\ell)(k-\ell-1)}{k(k-1)}$.

Proof. Let j be the least integer in $[k]$ such that $r_j \notin \hat{R}$. Fix a set u of size ℓ such that $1 \in u$ and $j \notin u$. Denote by v the set obtained from u by deleting 1 and inserting j. This defines a one to one correspondence (matching) between all such pair of (u, v). Moreover $r_v = r_u + r_j$ (notice that $r_1 = 0$). Then at least one of the element r_u or r_v is not in T. For otherwise $(r_v - r_u) \in \hat{R}$ implying $r_j \in \hat{R}$, which is a contradiction.

Therefore,

$$\text{Prob}_{u \in S_\ell}[r_u \in T \mid j \in u \text{ xor } 1 \in u] \leq \frac{1}{2}.$$

For any two indices i, j,

$$\text{Prob}_{u \in S_\ell}[i, j \in u \text{ or } i, j \notin u] = \frac{\ell(\ell-1) + (k-\ell)(k-\ell-1)}{k(k-1)} = p.$$

Thus,

$$\text{Prob}_{u \in S_\ell}[r_u \in T] \leq (1-p)/2 + p \leq (1+p)/2.$$

This completes the proof. ∎

Let $T = R_i$ in Lemma 3.5, where R_i is as defined in Lemma 3.4.

Suppose $f = 0$ is not an identity for the ring R. Then, using Lemma 3.5, it is easy to see that, for $u_1, u_2, \cdots, u_m$ picked uniformly at random from S_ℓ, $f(r_{u_1}, \cdots, r_{u_m})$ is non zero with non-negligible probability. This is analogous to [MN07, Lemma 2]. We include a proof for the sake of completeness.

Lemma 3.6. *Let $f(x_1, \cdots, x_m)$ be a multilinear polynomial (in commuting or noncommuting indeterminates) over R such that $f = 0$ is not an identity for the ring R. Then,*

$$\text{Prob}_{u_1, \cdots, u_m \in S_\ell}[f(r_{u_1}, \cdots, r_{u_m}) \neq 0] \geq \left(\frac{1-p}{2}\right)^m.$$

Proof. For $i \in [m]$, let R_i be the additive coset defined in Lemma 3.4. The proof is by simple induction on m. The proof for the base case of the induction (i.e for $m = 1$) follows easily from the definition of R_i and Lemma 3.5. By induction hypothesis assume that the result holds for all t-variate multilinear polynomials g such that $g = 0$ is not an identity for R with $t \leq m - 1$.

[2]Notice that in [MN07], the author consider the set of all ℓ tuples instead of subsets. This is important for them as they work in non abelian structure in general (where order matters). But we will be interested only over additive abelian structure of a ring and thus order does not matter for us.

Consider the given multilinear polynomial $f(x_1, x_2, \cdots, x_m)$. Then, by Lemma 3.4, R_m is a coset of an additive subgroup $\hat{R}_m$ inside R. Pick $u_m \in \mathrm{S}_\ell$ uniformly at random. If $f = 0$ is not an identity on R then by Lemma 3.5 we get $r_{u_m} \notin R_m$ with probability at least $\frac{1-p}{2}$. Let $g(x_1, x_2, \cdots, x_{m-1}) = f(x_1, \cdots, x_{m-1}, r_{u_m})$. Since $r_{u_m} \notin R_m$ with probability at least $\frac{1-p}{2}$, it follows that $g = 0$ is not an identity on R with probability at least $\frac{1-p}{2}$. Given that g is not an identity for R, by induction hypothesis we have that, $\text{Prob}_{u_1,\cdots,u_{m-1} \in S_\ell}[g(r_{u_1}, \cdots, r_{u_{m-1}}) \neq 0] \geq \left(\frac{1-p}{2}\right)^{m-1}$. Hence we get, $\text{Prob}_{u_1,\cdots,u_m \in S_\ell}[f(r_{u_1}, \cdots, r_{u_m}) \neq 0] \geq \left(\frac{1-p}{2}\right)^m$, which proves the lemma. ∎

We observe two simple consequences of Lemma 3.6. Notice that $\frac{1-p}{2} = \frac{\ell(k-\ell)}{k(k-1)}$. Letting $\ell = 1$ we get $\frac{1-p}{2} = 1/k$, and Lemma 3.6 implies that if $f = 0$ is not an identity for R then $f(a_1, \cdots, a_m) \neq 0$ for one of the k^m choices for the a_i from the generating set $\{r_1, \cdots, r_k\}$.

Letting $\ell = k/2$ in Lemma 3.6, we get $\frac{1-p}{2} \geq 1/4$. Hence we obtain the following randomized test which makes $4^m mk$ queries.

Corollary 3.7. *There is a randomized $4^m mk$ query algorithm for* MIT *with constant success probability, where f is m-variate and R is given by an additive generating set of size k. This can be seen as a generalization of Pak's $O(k)$ query randomized test for group commutativity.*

We use Lemma 3.6 to design our quantum algorithm. Technically, our quantum algorithm is similar to the one described in [MN07]. The Lemma 3.6 is used to guarantee that there will at least $\left(\frac{1-p}{2}\right)^m$ fraction of *marked points* in the space S_ℓ^m i.e. the points where f evaluates to non-zero. The underlying graph in our random walk is a Johnson Graph and our analysis require some simple modification of the analysis described in [MN07].

3.1.1. *Random walk on S_ℓ.* Our random walk can be described as a random walk over a graph $G = (V, E)$ defined as follows: The vertices of G are all possible ℓ subsets of $[k]$. Two vertices are connected by an edge whenever the corresponding sets differ by exactly one element. Notice that G is a connected $\ell(k-\ell)$-regular Johnson graph, with parameter $(k, \ell, \ell-1)$ [BCN89]. Let P be the normalized adjacency matrix of G with rows and columns are indexed by the subsets of $[k]$. Then $P_{XY} = 1/\ell(k-\ell)$ if $|X \cap Y| = \ell - 1$ and 0 otherwise. It is well known that the spectral gap δ of P ($\delta = 1 - \lambda$, where λ is the second largest eigenvalue of P) is $\Omega(1/\ell)$ for $\ell \leq k/2$ [BCN89]. Now we describe the random walk on G.

Let the current vertex is $u = \{u_1, u_2, \cdots, u_\ell\}$ and $r_u = r_{u_1} + r_{u_2} + \cdots + r_{u_\ell}$. With probability $1/2$ stay at u and with probability $1/2$ do the following: randomly pick $u_i \in u$ and $j \in [k] \setminus u$. Then move to vertex v such that v is obtained from u by removing u_i and inserting j. Compute r_v by simply subtracting r_{u_i} from r_u and adding r_j to it. That will only cost 2 oracle access. Staying in any vertex with probability $1/2$ ensures that the random walk is ergodic. So the stationary distribution of the random walk is always uniform. It is easy to see that the transition matrix of the random walk is $A = (I + P)/2$ where I is the identity matrix of suitable dimension. So the spectral gap of the transition matrix A is $\hat{\delta} = (1 - \lambda)/2 = \delta/2$.

The query complexity analysis is similar to the analysis of Magniez-Nayak. But to fit it with our requirement, we need some careful parameter setting. We include a brief self-contained proof.

Theorem 3.8. *Let R be a finite black-box ring given as an oracle and $f(x_1, \cdots, x_m)$ be a multilinear polynomial over R given as a black-box. Moreover let $\{r_1, \cdots, r_k\}$ be a given additive*

generating set for R. Then the quantum query complexity of testing whether f is an identity for R, is $O(m(1+\alpha)^{m/2}k^{\frac{m}{m+1}})$, assuming $k \geq (1+1/\alpha)^{m+1}$.

Proof. **Setup cost(S):** For the quantum walk step we need to start with an uniform distribution on S_ℓ^m. With each $u \in S_\ell$, we maintain a quantum register $|d_u\rangle$ that computes r_u. So we need to prepare the following state $|\Psi\rangle$:

$$|\Psi\rangle = \frac{1}{\sqrt{|S_\ell^m|}} \sum_{u_1,u_2,\cdots,u_m \in S_\ell^m} |u_1, r_{u_1}\rangle \otimes |u_2, r_{u_2}\rangle \otimes \cdots \otimes |u_m, r_{u_m}\rangle.$$

It is easy to see that to compute any r_{u_j}, we need $\ell - 1$ oracle access to the ring oracle. Since in each of m independent walk, quantum queries over all choices of u will be made in parallel (using quantum superposition), the total query cost for setup is $m(\ell-1)$.

Update cost(U): It is clear from the random walk described in the section 3.1.1, that the update cost over S_ℓ is only 2 oracle access. Thus for the random walk on S_ℓ^m which is just m independent random walks, one on each copy of S_ℓ, we need a total update cost $2m$.[3]

Checking cost(C): To check whether f is zero on a point during the walk, we simply query the oracle for f once.

Recall from Szegedy's result [Sze04] (as stated in Theorem 3.2), the total cost for query complexity is $Q = S + \frac{1}{\sqrt{\hat{\delta}\epsilon}}(U+C)$ where $\epsilon = \left(\frac{1-p}{2}\right)^m$ is the proportion of the marked elements and $\hat{\delta}$ is the spectral gap of the transition matrix A described in section 3.1.1. Combining together we get, $Q \leq m\left[(\ell-1) + \frac{3}{\sqrt{\hat{\delta}\epsilon}}\right]$. From the random walk described in the section 3.1.1, we know that $\hat{\delta} \geq \frac{1}{2\ell}$. Hence, $Q \leq m\left[(\ell-1) + \frac{3\sqrt{2\ell}}{\left(\frac{1-p}{2}\right)^{\frac{m}{2}}}\right]$. Notice that, $\frac{1-p}{2} = \frac{\ell}{k}\left(\frac{1-\frac{\ell}{k}}{1-\frac{1}{k}}\right)$. Substituting for $\frac{1-p}{2}$ we get, $Q \leq m\left[(\ell-1) + 3\sqrt{2}k^{m/2}\frac{1}{\ell^{\frac{m-1}{2}}\left(\frac{k-\ell}{k-1}\right)^{m/2}}\right]$. We will choose a suitably small $\alpha > 0$ so that $\frac{k-1}{k-\ell} < 1+\alpha$. Then we can upper bound Q as follows. $Q \leq m\left[(\ell-1) + 3\sqrt{2}\cdot(1+\alpha)^{m/2}k^{m/2}\frac{1}{\ell^{\frac{m-1}{2}}}\right]$. Now our goal is to minimize Q with respect to ℓ and α. For that we choose $\ell = k^t$ where we will fix t appropriately in the analysis. Substituting $\ell = k^t$ we get, $Q \leq m\left[(k^t-1) + 3\sqrt{2}\cdot(1+\alpha)^{m/2}t^{1/2}k^{\frac{m-(m-1)t}{2}}\right]$. Choosing $t = (m/(m+1))$, we can easily see that the query complexity of the algorithm is $O(m(1+\alpha)^{m/2}k^{\frac{m}{m+1}})$. Finally, recall that we need choose an $\alpha > 0$ so that $\frac{k-1}{k-\ell} \leq 1+\alpha$. Clearly, it suffices to choose α so that $(1+\alpha)\ell \leq \alpha k$. Letting $\ell = k^{m/m+1}$ we get the constraint $(1+1/\alpha)^{m+1} \leq k$ which is satisfied if $e^{(m+1)/\alpha} \leq k$. We can choose $\alpha = \frac{m+1}{\ln k}$. ∎

Remark 3.9. The choice of α in the above theorem shows some trade-offs in the query complexity between the parameters k and m. For constant m notice that this gives us an $O(k^{m/m+1})$ query complexity upper bound for the quantum algorithm, which is similar to the best known query upper bound for m-COLLISION [Amb07], when the problem instance is a function $f : [k] \to [k]$.

Generalized Multilinear Identity Testing (GMIT**):** We now consider a variant of the MIT problem, which we call GMIT (for generalized-MIT).

[3] In [MN07] the underlying group operation is not necessarily commutative (it is being tested for commutativity). Thus the update cost is more.

Let $f : R^m \to R$ be a black-box multilinear polynomial. Consider any *additive subgroup* A of the black-box ring R, given by a set of generators $r_1, r_2, \cdots, r_k$, so that $A = \{\sum_i \beta_i r_i \mid \beta_i \in \mathbb{Z}\}$. The GMIT$(R, A, f)$ problem is the following: test whether a black-box multilinear polynomial f is an identity for A. In other words, we need to test if $f(a_1, \cdots, a_m) = 0$ for all $a_i \in A$.

It is easy to observe that the quantum algorithm actually solves GMIT and the correctness proof and analysis given in Theorem 3.8 also hold for GMIT problem. We summarize this observation in the following theorem.

Theorem 3.10. *Let R be a black-box finite ring given by ring oracles and $A = \langle r_1, r_2, \cdots, r_k \rangle$ be an* additive subgroup *of R given by generators $r_i \in R$. Let $f(x_1, x_2, \cdots, x_m)$ be a black-box multilinear polynomial $f : R^m \to R$. Then there is a quantum algorithm with query complexity $O(m(1+\alpha)^{m/2} k^{\frac{m}{m+1}})$ for the* GMIT(R, A, f) *problem (assuming $k \geq (1 + 1/\alpha)^{m+1}$).*

4. Query Complexity Lower Bound

In this section we show that GMIT problem of multilinear identity testing for additive subgroups of a black-box ring (described in Section 3.1.1), is at least as hard as m-SPLIT COLLISION (again, m-SPLIT COLLISION problem is defined in Section 1). Also, the well-known m-COLLISION problem can be easily reduced to m-SPLIT COLLISION problem using a simple randomized reduction. In the following lemma, we briefly state the reduction.

Lemma 4.1. *There is a randomized reduction from* m-COLLISION *to* m-SPLIT COLLISION *with success probability close to e^{-m}.*

Proof. Let $f : [k] \to [k]$ be a 'yes' instance of m-COLLISION, and suppose $f^{-1}(i) = \{i_1, i_2, \cdots, i_m\}$. To reduce this instance to m-SPLIT COLLISION we pick a random m-partition $I_1, I_2, \cdots, I_m$ of the domain $[k]$ with each $|I_j| = k/m$. It is easy to see that, with probability close to e^{-m}, the set $\{i_1, i_2, \cdots, i_m\}$ will be a split collision for the function f. ∎

Consequently, showing a quantum lower bound of $\Omega(k^\alpha)$ for m-COLLISION will imply a quantum lower bound of $\Omega(k^\alpha/e^m)$ for m-SPLIT COLLISION. It will also show similar lower bound for GMIT because of our reduction.

If $f : [k] \to [k]$ is an instance of m-SPLIT COLLISION problem, then the classical randomized query complexity lower bound is $\Omega(k)$. This is observed in [MN07] for $m = 2$. Due to our reduction, we get similar randomized query complexity lower bound for GMIT.

Currently the best known quantum query complexity lower bound for m-COLLISION problem is $\Omega(k^{2/3})$ (in the case $m = 2$) [AS04]. Thus we obtain the same explicit lower bound for m-SPLIT COLLISION problem due to the random reduction from m-COLLISION to m-SPLIT COLLISION. It also implies quantum query complexity lower bound for GMIT.

Our reduction from m-SPLIT COLLISION to GMIT problem is based on some new automata theoretic ideas. We first describe necessary automata theoretic ideas those are useful for our reduction.

4.1. Automata theory background

We recall some standard automata theory notations (see, for example, [HU78]). Fix a finite automaton $A = (Q, \Sigma, \delta, q_0, q_f)$ which takes as input strings in Σ^*. Q is the set of states of A, Σ is the alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, and q_0 and q_f are the initial and final states respectively (throughout, we only consider automata with unique accepting states). For

each letter $b \in \Sigma$, let $\delta_b : Q \to Q$ be the function defined by: $\delta_b(q) = \delta(q, b)$. These functions generate a submonoid of the monoid of all functions from Q to Q. This is the transition monoid of the automaton A and is well-studied in automata theory: for example, see [Str94, page 55]. We now define the 0-1 matrix $M_b \in \mathbb{F}^{|Q|\times|Q|}$ as follows: $M_b(q, q') = 1$ if $\delta_b(q) = q'$, and 0 otherwise.

The matrix M_b is simply the adjacency matrix of the graph of the function δ_b. As the entries of M_b are only zeros and ones, we can consider M_b to be a matrix over any field $\mathbb{F}$.

Furthermore, for any $w = w_1 w_2 \cdots w_k \in \Sigma^*$, we define the matrix M_w to be the matrix product $M_{w_1} M_{w_2} \cdots M_{w_k}$. If w is the empty string, define M_w to be the identity matrix of dimension $|Q| \times |Q|$. For a string w, let δ_w denote the natural extension of the transition function to w. If w is the empty string, δ_w is simply the identity function. It is easy to check that: $M_w(q, q') = 1$ if $\delta_w(q) = q'$ and 0 otherwise. Thus, M_w is also a matrix of zeros and ones for any string w. Also, $M_w(q_0, q_f) = 1$ if and only if w is accepted by the automaton A. We now describe the reduction.

Theorem 4.2. *The* m-SPLIT COLLISION *problem reduces to* GMIT *problem for additive subgroups of black-box rings.*

Proof. An instance of m-SPLIT COLLISION is a function $f : [k] \to [k]$ given as an oracle, where we assume w.l.o.g. that $k = nm$. Divide $\{1, 2, \cdots, k\}$ into m intervals $I_1, I_2, \cdots, I_m$, each containing n consecutive points of $[k]$. Recall from Section1 that, f is said to have an m-split collision if for some $j \in [k]$ we have $|f^{-1}(j)| = m$ and $|f^{-1}(j) \cap I_i| = 1$ for each interval I_i.

Consider the alphabet $\Sigma = \{b, c, b_1, b_2, \cdots, b_m\}$. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, q_f)$ be a deterministic finite state automaton that accepts all strings $w \in \Sigma^*$ such that each $b_j, 1 \le j \le m$ occurs at least once in w. It is easy to see that such an automaton with a single final state q_f can be designed with total number of states $|Q| = 2^{O(m)} = t$. W.l.o.g. let the set of states Q be renamed as $\{1, 2, \cdots, t\}$, where 1 is the initial state and t is the final state.

For each letter $a \in \Sigma$, let M_a denote the $t \times t$ transition matrix for δ_a (as defined in Section 4.1). Since each M_a is a $t \times t$ 0-1 matrix, each M_a is in the ring $\mathcal{M}_t(\mathbb{F}_2)$ of $t \times t$ matrices with entries from the field $\mathbb{F}_2$. Let R denote the k-fold product ring $(\mathcal{M}_t(\mathbb{F}_2))^k$. Clearly, R is a finite ring (which is going to play the role of the black-box ring in our reduction). We now define an additive subgroup T of R, where we describe the generating set of T using the m-SPLIT COLLISION instance f.

For each index $i \in [k]$, define an k-tuple $T_i \in R$ as follows. If $i \neq f(i)$, then define $T_i[i] = M_b$, $T_i[f(i)] = M_{b_j}$ (where $i \in I_j$) and and for each index $s \notin \{i, f(i)\}$ define $T_i[s] = M_c$. For $i = f(i)$, define $T[i] = M_{b_j}$ ($i \in I_j$) and the rest of the entries as M_c. The additive subgroup of R that we consider is $T = \langle T_1, T_2, \cdots, T_k \rangle$ generated by the $T_i, 1 \le i \le k$.

Furthermore, define two $t \times t$ matrices A and B in $\mathcal{M}_t(\mathbb{F}_2)$ as follows. Let $A[1, 1] = 1$ and $A[u, \ell] = 0$ for $(u, \ell) \neq (1, 1)$. For the matrix B, let $B[t, 1] = 1$ and $B[u, \ell] = 0$ for $(u, \ell) \neq (t, 1)$.

Claim 1. Let $w = w_1 w_2 \cdots w_s \in \Sigma^*$ be any string. Then the automaton $\mathcal{A}$ defined above accepts w if and only if the matrix $AM_{w_1} M_{w_2} \cdots M_{w_s} B$ is nonzero.

Proof of Claim By definition of the matrices M_a, the $(1, t)^{th}$ entry of the product $M_{w_1} M_{w_2} \cdots M_{w_s}$ is 1 if and only if w is accepted by $\mathcal{A}$. By definition of the matrices A and B the claim follows immediately.

Now, consider the polynomial $P(x_1, x_2, \cdots, x_m)$ with coefficients from the matrix ring R defined as follows:

$$P(x_1, x_2, \cdots, x_m) = \bar{A} x_1 x_2 \cdots x_m \bar{B},$$

where $\bar{A} = (A, A, \ldots, A) \in R$ and $\bar{B} = (B, B, \cdots, B) \in R$ are k-tuples of A's and B's respectively. We claim that the multilinear polynomial $P(x_1, x_2, \cdots, x_m) = 0$ is an identity for the additive subgroup T if and only if f has no m-split collision.

Claim 2. $P(x_1, \cdots, x_m) = 0$ is an identity for the additive subgroup $T = \langle T_1, \cdots, T_k \rangle$ if and only if f has no m-split collision. In other words, GMIT(R, T, P) is an 'yes' instance if and only if f has no m-split collision.

Proof of Claim Suppose f has an m-split collision. Specifically, let $i_j \in I_j$ ($1 \leq j \leq m$ and $i_1 < i_2 < \cdots < i_m$) be indices such that $f(i_1) = \cdots = f(i_m) = \ell$. In the polynomial P, we substitute the indeterminate x_j by T_{i_j}.

Then $P(T_{i_1}, T_{i_2}, \cdots, T_{i_m}) = \bar{A}M\bar{B}$, where $M = T_{i_1} \cdots T_{i_m}$. M is a k-tuple of $t \times t$ matrices such that the ℓ^{th} component of M is $\prod_{j=1}^{m} M_{b_j}$ where $i_j \in I_j$. Since $b_{i_1} b_{i_2} \cdots b_{i_m} \in \Sigma^*$ is a length m-string containing all the b_j's it will be accepted by the automaton $\mathcal{A}$. Consequently, the $(q_0, q_f)^{th}$ entry of the matrix M, which is the $(1, t)^{th}$ entry, is 1 (as explained in Section 4.1). It follows that the $(1, 1)$ entry of the matrix AMB is 1. Hence $P = 0$ is not an identity over the additive subgroup T.

For the other direction, assume that f has no m-split collision. We need to show that $P = 0$ is an identity for the ring T. For any m elements $S_1, S_2, \cdots, S_m \in T$ consider $P(S_1, S_2, \cdots, S_m) = \bar{A}S_1S_2 \cdots S_m\bar{B}$. Since Each S_j is an $\mathbb{F}_2$-linear combination of the generators $T_1, \cdots, T_k$, it follows by distributivity in the ring R that $P(S_1, S_2, \cdots, S_m)$ is an $\mathbb{F}_2$-linear combination of terms of the form $P(T_{k_1}, T_{k_2}, \cdots, T_{k_m})$ for some m indices $k_1, \cdots, k_m \in [k]$. Thus, it suffices to show that $P(T_{k_1}, T_{k_2}, \cdots, T_{k_m}) = 0$.

Let $\hat{T} = T_{k_1} T_{k_2} \cdots T_{k_m}$. Then, for each $j \in [k]$ we have $\hat{T}[j] = T_{k_1}[j] T_{k_2}[j] \cdots T_{k_m}[j]$. Since f has no m-split collision, for each $j \in [N]$ the set of matrices $\{M_{b_1}, M_{b_2}, \cdots, M_{b_m}\}$ is *not* contained in the set $\{T_1[j], T_2[j], \cdots, T_k[j]\}$. Thus, $\hat{T}[j] = T_{k_1}[j] T_{k_2}[j] \cdots T_{k_m}[j]$ is a product of matrices $M_{w_1} M_{w_2} \cdots M_{w_m}$ for a word $w = w_1 w_2 \cdots w_m$ that is not accepted by $\mathcal{A}$. It follows from the previous claim that $A\hat{T}[j]B = 0$. Hence $P(T_{k_1}, T_{k_2}, \cdots, T_{k_m}) = 0$ which completes the proof. ∎

In Section 3.1, we have already shown a quantum algorithm of query complexity $O(k^{\frac{m}{m+1}})$ for MIT (m is a constant). This bound holds as well for GMIT. We conclude this section by showing that any algorithm of query complexity $q(k, m)$ (q is any function) for GMIT will give an algorithm of similar query complexity for m-COLLISION problem. In particular an algorithm for GMIT of query complexity $k^{o(m/m+1)}$ will improve the best known algorithm for m-COLLISION problem due to Ambainis [Amb07]. The following corollary is an easy consequence of Theorem 4.2.

Corollary 4.3. *Let $f : [k] \to [k]$ be an instance of* m-SPLIT COLLISION *problem and* GMIT(R, T, P) *be an instance of* GMIT *problem, where the multilinear polynomial $P : R^m \to R$ and T is an additive subgroup of G given by k generators. Then, if we have a quantum algorithm of query complexity $q(k, m)$ for* GMIT *problem, we will have a quantum algorithm for* m-SPLIT COLLISION *with query complexity $O(q(k, m))$.*

Proof. Let $\mathcal{A}$ be an algorithm for GMIT with quantum query complexity $q(k, m)$. Given an instance of m-SPLIT COLLISION, the generators for the additive subgroup T is indexed by $1, 2, \cdots, k$ (as defined in the proof of Theorem 4.2). Also, define the polynomial $P(x_1, x_2, \cdots, x_m)$ So the inputs of our GMIT problem are $1, 2, \cdots, k$ and P. Using the algorithm $\mathcal{A}$, we define another algorithm $\mathcal{A}'$ which does the following. When $i \in [k]$ is invoked by $\mathcal{A}$ for the ring operation, the algorithm $\mathcal{A}'$ constructs the generator T_i by making only one query to the oracle for f. One more query to the f-oracle is required to erase the output. Moreover, if $\mathcal{A}$ wants to check whether the output of the ring operation is a valid generator (say T_j for some j), then also $\mathcal{A}'$ uses just two queries to the oracle of f. Thus we have an algorithm $\mathcal{A}'$ for m-SPLIT COLLISION with query complexity $4q(k)$. ∎

Recall that the best known lower bound for m-SPLIT COLLISION problem is $\Omega(k^{2/3})$. Then, combining Theorem 4.2 and Corollary 4.3, we get $\Omega(k^{2/3})$ quantum query lower bound for GMIT problem.

Acknowledgement

We thank Ashwin Nayak for comments and suggestions. We are grateful to the anonymous STACS'09 referees for useful comments.

References

[ADM06] Vikraman Arvind, Bireswar Das, and Partha Mukhopadhyay. The complexity of black-box ring problems. In *COCOON*, pages 126–135, 2006.

[Amb07] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, 2007.

[AMS08] Vikraman Arvind, Partha Mukhopadhyay, and Srikanth Srinivasan. New results on noncommutative and commutative polynomial identity testing. In *IEEE Conference on Computational Complexity*, pages 268–279, 2008.

[AS04] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, 2004.

[BCN89] A.E. Brouwer, A.M. Cohen, and A.N. Neumaier. Distance Regular Graphs. *Springer-Verlag*, pages 255–256, 1989.

[HU78] John E. Hopcroft and Jeffrey D. Ullman. Introduction to Automata Theory, Languages and Computation. *Addison-Wesley*, 1978.

[MN07] Frederic Magniez and Ashwin Nayak. Quantum complexity of testing group commutativity. *Algorithmica*, 48(3):221–232, 2007.

[MNRS07] Frederic Magniez, Ashwin Nayak, Jeremie Roland, and Miklos Santha. Search via quantum walk. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 575–584, 2007.

[Pak00] Igor Pak. Testing commutativity of a group and the power of randomization. *Electronic Version at: http://www-math.mit.edu/pak/research.html*, 2000.

[Str94] Howard Straubing. Finite Automata, Formal Logic, and Circuit Complexity. *Progress in Theoretical Computer Science, Birkhser*, 1994.

[Sze04] Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–41, 2004.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 99–110
www.stacs-conf.org

AN ORDER ON SETS OF TILINGS CORRESPONDING TO AN ORDER ON LANGUAGES

NATHALIE AUBRUN [1] AND MATHIEU SABLIK [2]

[1] Institut Gaspard Monge, Université Paris-Est Marne-la-Vallée,
77454 Marne-la-Valle Cedex 2, France.
E-mail address: nathalie.aubrun@univ-mlv.fr

[2] Laboratoire d'Analyse, Topologie, Probabilité, Université de Provence,
39, rue F. Joliot Curie, 13453 Marseille Cedex 13, France.
E-mail address: sablik@cmi.univ-mrs.fr

Abstract. Traditionally a tiling is defined with a finite number of finite forbidden patterns. We can generalize this notion considering any set of patterns. Generalized tilings defined in this way can be studied with a dynamical point of view, leading to the notion of subshift. In this article we establish a correspondence between an order on subshifts based on dynamical transformations on them and an order on languages of forbidden patterns based on computability properties.

Introduction

Given a finite set of tiles $\mathcal{A}$ and a finite set of forbidden patterns P, a d-dimensional tiling is an element of $\mathcal{A}^{\mathbb{Z}^d}$ where the local conditions imposed by P are satisfied at every point of $\mathbb{Z}^d$. This basic model captures geometrical aspect of computation [Ber66, Rob71, Han74]. To establish structural properties of tilings, it is interesting to study the set of tilings which satisfy the conditions imposed by P [BDJ08].

It is easy to generalize the usual notion of tiling considering an infinite set of forbidden patterns. A set of generalized tilings can be studied with a dynamical point of view with the notion of subshift [LM95, Kit98]. In this theory, a set of usual tilings corresponds to a subshift of finite type.

In dimension 1, the class of subshifts of finite type is well understood. In particular, the language of a subshift of finite type is given by a local automaton [Bea93]. Given this result, it is natural to characterize subshifts whith a language given by a finite automaton. It is the class of sofic subshifts, which can all be obtained as a factor of a subshift of finite type [LM95]. Thus, each sofic subshift is obtained by a dynamical transformation of a subshift of finite type.

Multidimensional subshifts of finite type are not well understood. For example, it is not easy to describe their languages. Moreover, in addition to factors, there exist other types of dynamical transformations on multidimensional subshift: the sub-action of a d-dimensional

1998 ACM Subject Classification: G.2.m.
Key words and phrases: tiling, subshift, Turing machine with oracle, subdynamics.

tiling consists in taking the restriction of a tiling to a subgroup of $\mathbb{Z}^d$. Hochman showed that every d-dimensional subshift whose set of forbidden patterns is recursively enumerable can be obtained by sub-action and factor of a $d+2$-subshift of finite type [Hoc07].

This result suggests that a subshift can simulate another one, where the notion of simulation is given by operations on subshifts inspired by the dynamical theory. This involves different orders depending on the operations which are considered. In this paper, we present five types of operations: product, factor, finite type, sub-action and superposition. It is possible to formulate classic results with this formalism. Our main result (Theorem 4.2) establishes a correspondence between an order on subshifts based on dynamical transformations on them and an order on languages of forbidden patterns based on computability properties.

The paper is organized as follows: Section 1 is devoted to introduce the concepts of tiling and subshift. In Section 2, we present several operations on subshifts which allow to define the notion of simulation of a subshift by another one. Then, in Section 3, we define an important tool to define runs of a Turing machine with a sofic subshift. This tool is used to prove our main result in the last Section.

1. Definitions

1.1. Generalized tilings

Let $\mathcal{A}$ be a finite alphabet and d be a positive integer. A *configuration* x is an element of $\mathcal{A}^{\mathbb{Z}^d}$. Let $\mathbb{S}$ be a finite subset of $\mathbb{Z}^d$. Denote $x_{\mathbb{S}}$ the *restriction* of x to $\mathbb{S}$. A *pattern* is an element $p \in \mathcal{A}^{\mathbb{S}}$ and $\mathbb{S}$ is the *support* of p, which is denoted by $\mathrm{supp}(p)$. For all $n \in \mathbb{N}$, we call $\mathbb{S}_n^d = [-n;n]^d$ the *elementary support* of size n. A pattern with support $\mathbb{S}_n^d$ is an *elementary pattern*. We denote by $\mathcal{E}_{\mathcal{A}}^d = \cup_{n\in\mathbb{N}} \mathcal{A}^{[-n;n]^d}$ the set of d-dimensional elementary patterns. A *d-dimensional language* $\mathcal{L}$ is a subset of $\mathcal{E}_{\mathcal{A}}^d$. A pattern p of support $\mathbb{S} \subset \mathbb{Z}^d$ *appears* in a configuration x if there exists $i \in \mathbb{Z}^d$ such that for all $j \in \mathbb{S}$, $p_j = x_{i+j}$, we note $p \sqsubset x$.

Definition 1.1. A *tile set* is a tuple $\tau = (\mathcal{A}, P)$ were P is a subset of $\mathcal{E}_{\mathcal{A}}^d$ called the *set of forbidden patterns*.

A *generalized tiling* by τ is a configuration x such that for all $p \in P$, p does not appear in x. We denote by $\mathbf{T}_\tau$ the set of generalized tilings by τ. If there is no ambiguity on the alphabet, we just denote it by $\mathbf{T}_P$.

Remark 1.2. If P is finite, it is equivalent to define a generalized tiling by allowed patterns or forbidden patterns, the latter being the usual definition of tiling.

1.2. Dynamical point of view : subshifts

One can define a topology on $\mathcal{A}^{\mathbb{Z}^d}$ by endowing $\mathcal{A}$ with the discrete topology, and considering the product topology on $\mathcal{A}^{\mathbb{Z}^d}$. For this topology, $\mathcal{A}^{\mathbb{Z}^d}$ is a compact metric space on which $\mathbb{Z}^d$ acts by translation via σ defined by:

$$\begin{array}{rccl} \sigma_{\mathcal{A}}^i : & \mathcal{A}^{\mathbb{Z}^d} & \longrightarrow & \mathcal{A}^{\mathbb{Z}^d} \\ & x & \longmapsto & \sigma_{\mathcal{A}}^i(x) \quad \text{such that } \sigma_{\mathcal{A}}^i(x)_u = x_{i+u}\ \forall u \in \mathbb{Z}^d. \end{array}$$

for all i in $\mathbb{Z}^d$. This action is called the shift.

Definition 1.3. A d-dimensional subshift on the alphabet $\mathcal{A}$ is a closed and σ-invariant subset of $\mathcal{A}^{\mathbb{Z}^d}$. We denote by $\mathcal{S}$ (resp. $\mathcal{S}_d$, $\mathcal{S}_{\leq d}$) the set of all subshifts (resp. d-dimensional subshifts, d'-dimensional subshifts with $d' \leq d$).

Let $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be a subshift. Denote $\mathcal{L}_n(\mathbf{T}) \subseteq \mathcal{A}^{[-n;n]^d}$ the set of elementary patterns of size n which appear in some element of $\mathbf{T}$, and $\mathcal{L}(\mathbf{T}) = \cup_{n\in\mathbb{N}}\mathcal{L}_n(\mathbf{T})$ the *language* of $\mathbf{T}$ which is the set of elementary patterns which appear in some element of $\mathbf{T}$.

It is also usual to study a subshift as a dynamical system [LM95, Kit98], the next proposition shows the link between both notions.

Proposition 1.4. *The set $\mathbf{T} \subset \mathcal{A}^{\mathbb{Z}^d}$ is a subshift if and only if $\mathbf{T} = \mathbf{T}_{\mathcal{L}(\mathbf{T})^c}$ where $\mathcal{L}(\mathbf{T})^c$ is the complement of $\mathcal{L}(\mathbf{T})$ in $\mathcal{E}^d_{\mathcal{A}}$.*

Definition 1.5. Let $\mathcal{A}$ be a finite alphabet and $\mathbf{T} \subset \mathcal{A}^{\mathbb{Z}^d}$ be a subshift.

The subshift $\mathcal{A}^{\mathbb{Z}^d}$ is the *full-shift* associated to $\mathcal{A}$. Denote $\mathcal{FS}$ the set of all full-shifts.

If there exists a finite set $P \subseteq \mathcal{E}^d_{\mathcal{A}}$ such that $\mathbf{T} = \mathbf{T}_P$ then $\mathbf{T}$ is a *subshift of finite type*. Denote $\mathcal{SFT}$ the set of all subshifts of finite type. Subshifts of finite type correspond to the usual notion of tiling.

If there exists a recursively enumerable set $P \subseteq \mathcal{E}^d_{\mathcal{A}}$ such that $\mathbf{T} = \mathbf{T}_P$ then $\mathbf{T}$ is a *recursive enumerable subshift*. Denote $\mathcal{RE}$ the set of all recursive enumerable subshifts.

2. Operations on tilings

2.1. Simulation of a tiling by another one

An *operation op* on subshifts transforms a subshift or a pair of subshifts into another one; it is a function $op : \mathcal{S} \to \mathcal{S}$ or $op : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$. We remark that a subshift $\mathbf{T}$ (resp. a pair of subshifts $(\mathbf{T}', \mathbf{T}'')$) and the image by an operation $op(\mathbf{T})$ (resp. $op(\mathbf{T}', \mathbf{T}'')$) do not necessary have the same alphabet or dimension. An operation can depend on a parameter.

Let Op be a set of operations on subshifts. Let $\mathcal{U} \subset \mathcal{S}$ be a set of subshifts. We define the *closure* of $\mathcal{U}$ under a set of operations Op, denoted by $\mathcal{C}l_{Op}(\mathcal{U})$, as the smallest set stable by Op which contains $\mathcal{U}$.

We say that a subshift $\mathbf{T}$ *simulates* a subshift $\mathbf{T}'$ by Op if $\mathbf{T}' \in \mathcal{C}l_{Op}(\mathbf{T})$. Thus there exists a finite sequence of operations chosen among Op, that transforms $\mathbf{T}$ into $\mathbf{T}'$. We note it by $\mathbf{T}' \leq_{Op} \mathbf{T}$. We remark that $\mathcal{C}l_{Op}(\mathbf{T}) = \{\mathbf{T}' \ : \ \mathbf{T}' \leq_{Op} \mathbf{T}\}$.

2.2. Local transformations

We describe three operations that modify locally the subshift.

• **Product P:**

Let $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ and $\mathbf{T}' \subseteq \mathcal{B}^{\mathbb{Z}^d}$ be two subshifts of the same dimension, define:

$$\phi_P(\mathbf{T}, \mathbf{T}') = \mathbf{T} \times \mathbf{T}' \subseteq (\mathcal{A} \times \mathcal{B})^{\mathbb{Z}^d}.$$

One has $\mathcal{C}l_P(\mathcal{FS}) = \mathcal{FS}$ and $\mathcal{C}l_P(\mathcal{SFT}) = \mathcal{SFT}$.

• **Finite type FT:**

These operations consist in adding a finite number of forbidden patterns to the initial subshift. Formally, let $\mathcal{A}$ be an alphabet, $P \subseteq \mathcal{E}^d_{\mathcal{A}}$ be a finite subset and let $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be a subshift. By Proposition 1.4, there exists P' such that $\mathbf{T} = \mathbf{T}_{P'}$. Define:

$$\phi_{FT}(P, \mathbf{T}) = \mathbf{T}_{P\cup P'}.$$

If P and $\mathbf{T}$ have not the same alphabet or the same dimension, put $\phi_{FT}(P, \mathbf{T}) = \mathbf{T}$. We remark that $\phi_{FT}(P, \mathbf{T})$ could be empty if P prohibits too many patterns. By FT, one lists all operations on subshifts which are obtained by ϕ_{FT}.

By definition of subshift of finite type, one has $\mathcal{C}l_{FT}(\mathcal{FS}) = \mathcal{SFT}$.

• **Factor F:**

These operations allow to change the alphabet of a subshift by local modifications. Let $\mathcal{A}$ and $\mathcal{B}$ be two finite alphabets. A *morphism* $\pi : \mathcal{A}^{\mathbb{Z}^d} \to \mathcal{B}^{\mathbb{Z}^d}$ is a continuous function which commutes with the shift action (i.e. $\sigma^i \circ \pi = \pi \circ \sigma^i$ for all $i \in \mathbb{Z}^d$). In fact, such a function can be defined locally [Hed69]: that is to say, there exists $\mathbb{U} \subset \mathbb{Z}^d$ finite, called *neighborhood*, and $\overline{\pi} : \mathcal{A}^{\mathbb{U}} \to \mathcal{B}$, called *local function*, such that $\pi(x)_i = \overline{\pi}(x_{i+\mathbb{U}})$ for all $i \in \mathbb{Z}^d$. Let $\mathbf{T}$ be a subshift, define:

$$\phi_F(\pi, \mathbf{T}) = \pi(\mathbf{T}).$$

If the domain of π and $\mathbf{T}$ do not have the same alphabet or the same dimension, put $\phi_F(\pi, \mathbf{T}) = \mathbf{T}$. By F, one lists all operations on subshifts which are obtained by ϕ_F.

One verifies that $\mathcal{C}l_F(\mathcal{SFT}) \neq \mathcal{SFT}$.

Definition 2.1. A sofic subshift is a factor of a subshift of finite type. Thus, the set of sofic subshifts is $\mathcal{S}ofic = \mathcal{C}l_F(\mathcal{SFT})$.

2.3. Transformation on the group of the action

We describe two operations that modify the group on which the subshift is defined, thus we change the dimension of the subshift.

• **Sub-action SA:**

These operations allow to take the restriction of a subshift of $\mathcal{A}^{\mathbb{Z}^d}$ according to a subgroup of $\mathbb{Z}^d$. Let $\mathbb{G}$ be a sub-group of $\mathbb{Z}^d$ generated by $u_1, u_2, \ldots, u_{d'}$ ($d' \leq d$). Let $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be a subshift, define:

$$\phi_{SA}(\mathbb{G}, \mathbf{T}) = \left\{ y \in \mathcal{A}^{\mathbb{Z}^{d'}} \ : \ \exists x \in \mathbf{T} \text{ such that } \forall i_1, \ldots, i_{d'} \in \mathbb{Z}^{d'}, y_{i_1,\ldots,i_{d'}} = x_{i_1 u_1 + \cdots + i_{d'} u_{d'}} \right\}.$$

It is easy to prove that $\phi_{SA}(\mathbb{G}, \mathbf{T})$ is a subshift of $\mathcal{A}^{\mathbb{Z}^{d'}}$. If $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ and $\mathbb{G}$ is not a subgroup of $\mathbb{Z}^d$, put $\phi_{SA}(\mathbb{G}, \mathbf{T}) = \mathbf{T}$. By SA, one lists all operations on subshifts which are obtained by ϕ_{SA}.

One verifies that $\mathcal{C}l_{SA}(\mathcal{SFT}) \neq \mathcal{SFT}$ and $\mathcal{C}l_{SA}(\mathcal{SFT}) \neq \mathcal{S}ofic$.

Theorem 2.2. $\mathcal{C}l_{SA}(\mathcal{RE}) = \mathcal{RE}$.

• **Superposition SP:**

These operations increase the dimension of a subshift by a superposition of the initial subshift. Let $d, d' \in \mathbb{N}^*$. Let $\mathbb{G}$ and $\mathbb{G}'$ be two subgroups of $\mathbb{Z}^{d+d'}$ such that $\mathbb{G}$ is isomorphic to $\mathbb{Z}^d$ and $\mathbb{G} \oplus \mathbb{G}' = \mathbb{Z}^{d+d'}$. Let $\mathbf{T} \subseteq \mathcal{A}^{Z^d}$ be a subshift, define:

$$\phi_{SP}(\mathbb{G}, \mathbb{G}', \mathbf{T}) = \left\{ x \in \mathcal{A}^{\mathbb{Z}^{d+d'}} \ : \ \forall i \in \mathbb{G}', x_{i+\mathbb{G}} \in \mathbf{T} \right\}.$$

If $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ and $\mathbb{G}$ is not isomorphic to $\mathbb{Z}^d$ or $\mathbb{G} \oplus \mathbb{G}' \neq \mathbb{Z}^{d+d'}$, put $\phi_{SP}(\mathbb{G}, \mathbb{G}', \mathbf{T}) = \mathbf{T}$. By SP, one lists all operations on subshifts which are obtained by ϕ_{SP}.

It is easy to verify that $\mathcal{C}l_{SP}(\mathcal{SFT}) = \mathcal{SFT}$.

With this formalism, the result of M. Hochman [Hoc07] can be written:

$$\mathcal{C}l_{F,SA}(\mathcal{SFT}) = \mathcal{RE}.$$

More precisely, he proves that $\mathcal{C}l_{F,SA}(\mathcal{SFT} \cap \mathcal{S}_{d+2}) \cap \mathcal{S}_{\leq d} = \mathcal{RE} \cap \mathcal{S}_{\leq d}$.

3. Simulation of Turing machines by subshifts

A Turing machine is a model of calculation defined by local rules. It seems natural to represent the runs of a machine by a 2-dimensional subshift: one dimension representing the tape and the other time evolution. But the main problem is that in general the Turing machine uses a finite part of the space-time diagram which is represented by the subshift. Robinson [Rob71] proposes a self-similar structure to construct an aperiodic subshift of finite type of dimension 2. In fact, it is also possible to use a general construction with substitutions due to Mozes [Moz89]. This construction allows to give to the machine finite spaces on which it calculates independently. The problem is that we cannot control the entry of the Turing machine in view to recognize a configuration of a subshift. To obtain this property, Hochman [Hoc07] uses similar tools to construct a sofic subshift of dimension 3 in order to to prove that $\mathcal{C}l_{F,S!A}(\mathcal{SFT}) = \mathcal{RE}$. In this Section, we present a similar construction which is used to prove our main result in Section 4.

3.1. Substitution tilings

Let $\mathcal{A}$ be a finite alphabet. A *substitution* is a function $s : \mathcal{A} \to \mathcal{A}^{\mathbb{U}_k}$ where $\mathbb{U}_k = [1;k] \times [1;k]$. We naturally extend s to a function $s^n : \mathcal{A}^{\mathbb{U}_n} \to \mathcal{A}^{\mathbb{U}_{nk}}$ by identifying $\mathcal{A}^{\mathbb{U}_{nk}}$ with $(\mathcal{A}^{\mathbb{U}_k})^{\mathbb{U}_n}$. Starting from a letter placed in $(1,1) \in \mathbb{Z}^2$ and applying successively $s, s^k, \dots, s^{k^{n-1}}$ we obtain a sequence of patterns in $\mathcal{A}^{\mathbb{U}_{k^i}}$ for $i \in \{0, \dots, n\}$. Such patterns are called *s-patterns*.

Definition 3.1. The subshift $\mathbf{S}_s$ defined by the substitution s is

$$\mathbf{S}_s = \left\{ x \in \mathcal{A}^{\mathbb{Z}^2} \ : \ \text{every finite pattern of } x \text{ appears in a } s\text{-pattern} \right\}.$$

3.2. A framework for Turing machines

We now describe a family of substitutions s_n defined on the alphabet $\{\circ, \bullet\}$, which are used by M. Hochman [Hoc07] to prove $\mathcal{C}l_{F,SA}(\mathcal{SFT}) = \mathcal{RE}$. For every integer n the substitution s_n is given by :

$$\circ \longmapsto \begin{array}{cccc} \circ & \dots & \circ & \circ \\ \vdots & \iddots & \bullet & \circ \\ \circ & \iddots & \iddots & \vdots \\ \bullet & \circ & \dots & \circ \end{array} \quad \text{and} \quad \bullet \longmapsto \begin{array}{cccc} \circ & \dots & \circ & \bullet \\ \vdots & \iddots & \bullet & \circ \\ \circ & \iddots & \iddots & \vdots \\ \bullet & \circ & \dots & \circ \end{array}$$

where the patterns are of size $n \times n$. Let $\mathbf{S}_n$ be the tiling defined by substitution s_n.

These substitutions have good properties, in particular they are unique derivation substitutions and for this reason they verify [Moz89]; one obtains:

Proposition 3.2. *For every integer n, there exists a SFT $\tilde{\mathbf{S}}_n$ and a letter-to-letter morphism π_n such that $\mathbf{S}_n = \pi_n(\tilde{\mathbf{S}}_n)$.*

Definition 3.3. If $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^2}$ is a subshift, we define $\mathbf{T}^{(\uparrow)}$ by :

$$\mathbf{T}^{(\uparrow)} = \left\{ x \in \mathcal{A}^{\mathbb{Z}^2} \ : \ \exists y \in \mathbf{T}, \forall (i,j) \in \mathbb{Z}^2, x_{(i,j)} = y_{(i,j-i)} \right\}.$$

Notice that if $\mathbf{T}$ is an SFT, then $\mathbf{T}^{(\uparrow)}$ is also an SFT (just shift the forbidden patterns of $\mathbf{T}$ to get those of $\mathbf{T}^{(\uparrow)}$).

We now work on the space $\mathbb{Z}^3 = \mathbb{Z}e_1 \oplus \mathbb{Z}e_2 \oplus \mathbb{Z}e_3$ and we construct the SFT $\mathbf{W}_2$, $\mathbf{W}_3$ and $\mathbf{W}_5 \subseteq \{\circ, \bullet\}^{\mathbb{Z}^3}$ defined by :

$$x \in \mathbf{W}_2 \iff \begin{cases} \forall k \in \mathbb{Z}, x_{|\mathbb{Z}^2 \times \{k\}} \in \mathbf{S}_2^{(\uparrow)} \\ \forall u \in \mathbb{Z}^3, x_u = x_{u+e_3} \ (*) \end{cases} \qquad x \in \mathbf{W}_3 \iff \begin{cases} \forall j \in \mathbb{Z}, x_{|\mathbb{Z} \times \{j\} \times \mathbb{Z}} \in \mathbf{S}_3^{(\uparrow)} \\ \forall u \in \mathbb{Z}^3, x_u = x_{u+e_2} \ (**) \end{cases}$$

$$x \in \mathbf{W}_5 \iff \begin{cases} \forall k \in \mathbb{Z}, x_{|\mathbb{Z}^2 \times \{k\}} \in \mathbf{S}_5^{(\uparrow)} \\ \forall u \in \mathbb{Z}^3, x_u = x_{u+e_3} \ (***) \end{cases}$$

Let x be a configuration of the subshift $\mathbf{W}_2 \times \mathbf{W}_3 \times \mathbf{W}_5 \subseteq (\{\circ, \bullet\}^3)^{\mathbb{Z}^3}$. If we focus on the subshift $\mathbf{W}_3 \times \mathbf{W}_5$, we can see rectangles whose corners are defined by the letter $(\bullet, \bullet)$ of $\{\circ, \bullet\}^2$. These rectangles of size $5^n \times 3^m$ are spaces of calculation on which the Turing machine runs independently. Moreover the information brought by $\mathbf{W}_2$ gives the size of the entry pattern p on each rectangle : scanning the base of a rectangle from left to right, the entry word is located between the left corner and the first symbol $\bullet$ due to $\mathbf{W}_2$ that occurs. This results are resumed in Proposition 3.4.

Proposition 3.4. *The product $\mathbf{W}_2 \times \mathbf{W}_3 \times \mathbf{W}_5$ is a partition of the space into rectangles, in which each plane $\{i\} \times \mathbb{Z}^2$ is paved by rectangles of same width and height. Moreover if there is a $5^m \times 3^p$-rectangle in $(i,j,k) \in \mathbb{Z}^3$ with entry of size 2^n, then there exists i' and i'' such that there exists a $5^{m+1} \times 3^p$-rectangle in (i',j,k) and a $5^m \times 3^{p+1}$-rectangle in (i'',j,k) both with entry of size 2^n.*

This result will be used in Section 4.2.2 to prove that, thanks to these arbitrary large rectangles, one can simulate a calculation with an arbitrary number of steps.

3.3. A 2-dimensional sofic subshift

We now explain how we can use the previously constructed framework to simulate a Turing machine by a subshift. First we recall the formal definition of a Turing machine.

Definition 3.5. Let $\mathcal{M} = (Q, \mathcal{A}, \Gamma, \sharp, q_0, \delta, Q_F)$ be a Turing machine, where :

- Q is a finite set of states; $q_0 \in Q$ is the initial state;
- $\mathcal{A}$ are Γ are two finite alphabets such that $\mathcal{A} \subsetneq \Gamma$;
- $\sharp \notin \Gamma$ is the blank symbol;
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \cdot, \rightarrow\}$ is the transition function;
- $F \subset Q_F$ is the set of final states.

We can describe its behaviour with a set of 2-dimensional patterns. First dimension stands for the tape and second dimension for time evolution. For example the rule $\delta(q_1, x) = (q_2, y, \leftarrow)$ will be coded by :

(q_2, z)	y	z'
z	(q_1, x)	z'

Denote by $P_{\mathcal{M}}$ the set of forbidden patterns constructed according to the rules of $\mathcal{M}$. One can consider the subshift of finite type $\mathbf{T}_{P_{\mathcal{M}}}$ where each local pattern corresponds to calculations of the machine $\mathcal{M}$. Then thanks to a product operation we superimpose these calculations on the framework, with the following finite conditions :

- condition **Init** : to copy out the entry word ;
- condition **Head** : the initial state q_0 appears on every rectangle bottom left corner and only here;
- condition **Stop** : when a side of a rectangle is reached by the head of the machine, the calculation stops and if necessary the tape content is just copied out until the top of the rectangle;
- condition **Final** : when a final state is reached, the tape content is just copied out for next steps of calculation until the top of the rectangle.

Define $\mathbf{T}_{\mathcal{M}}$ the subshift:

$$\mathbf{T}_{\mathcal{M}} = \phi_{FT}\left(\{\mathbf{Init}, \mathbf{Head}, \mathbf{Stop}, \mathbf{Final}\}, \mathcal{A}^{\mathbb{Z}^3} \times (\mathbf{W}_2 \times \mathbf{W}_3 \times \mathbf{W}_5) \times \phi_{SP}(\mathbb{Z}e_2 \oplus \mathbb{Z}e_3, \mathbb{Z}e_1, \mathbf{T}_{P_{\mathcal{M}}})\right).$$

By stability of the class of subshifts of finite type by SP, $\mathbf{T}_{\mathcal{M}}$ is a subshift of finite type up to a letter-to-letter morphism; thus $\mathbf{T}_{\mathcal{M}} \in \mathcal{S}ofic$. For all $i \in \mathbb{Z}$, in the plane $\{i\} \times \mathbb{Z}^2$, it is possible to find rectangles of size $5^m \times 3^p$ arbitrary large and an entry of size 2^n also arbitrarily large. On each rectangle, thanks to the conditions $P_{\mathcal{M}}$, we can observe the evolution of the Turing machine $\mathcal{M}$.

Remark 3.6. The construction described here only works for usual Turing machines. In Section 4.2.2 we explain how to add finite conditions on the subshift $\mathbf{T}_{\mathcal{M}}$ if $\mathcal{M}$ is a Turing machine with oracle.

4. Study of the semi-order $\leq_{P,F,FT,SA,SP}$

In this section we focus on the five operations described previously. Our aim is to study the semi-order $\leq_{P,F,FT,SA,SP}$.

4.1. A semi-order on languages

A Turing machine with *semi-oracle* is a usual machine with a special state $q_?$ and an oracle tape. The behaviour of a Turing machine with semi-oracle $\mathcal{L}$, where $\mathcal{L}$ is a language, is the following : the machine reads an entry pattern p and writes a pattern on the oracle tape, until the state $q_?$ is reached. If the pattern written on the oracle tape is in $\mathcal{L}$ then the machine stops, else it keeps on calculating.

We define a semi-order on languages :

$$\mathcal{L} \preceq \mathcal{L}' \iff \exists \mathcal{M}^{\mathcal{L}'} \text{ a Turing machine with semi-oracle } \mathcal{L}' \text{ such that } dom(\mathcal{M}^{\mathcal{L}'}) = \mathcal{L},$$

where $dom(\mathcal{M})$ is the domain of the machine $\mathcal{M}$, that is to say the set of entry words on which $\mathcal{M}$ stops. We refer to [RJ87] for definitions and properties of similar semi-orders on languages based on computability.

Proposition 4.1. *$\preceq$ is a semi-order.*

Consider the equivalence relation $\mathcal{L} \approx \mathcal{L}'$ if and only if $\mathcal{L} \preceq \mathcal{L}'$ and $\mathcal{L}' \preceq \mathcal{L}$. This equivalence relation defines classes of languages, and we can compare them within the semi-order. For instance, the class of recursively enumerable languages is the smallest for this semi-order. We have $\emptyset \approx \mathcal{L}$ for every recursively enumerable language $\mathcal{L}$.

4.2. Closure theorem:

The semi-order on languages defined by semi-oracle Turing machines corresponds to a semi-order on subshifts:

Theorem 4.2. *Let* $\mathbf{T}$ *be a subshift, one has:*

$$\mathcal{C}l_{P,F,SA,SP,FT}(\mathbf{T}) = \{\mathbf{T}_{\mathcal{L}} : \mathcal{L} \preceq \mathcal{L}(\mathbf{T})^c\}.$$

Or equivalently, if $\mathbf{T}'$ *and* $\mathbf{T}''$ *are two subshifts of dimension* d' *and* d''*, one has:*

$$\mathbf{T}' \leq_{P,F,FT,SA,SP} \mathbf{T}'' \iff \mathcal{L}(\mathbf{T}')^c \preceq \mathcal{L}(\mathbf{T}'')^c.$$

4.2.1. *Direct inclusion.* Put $\mathcal{L} = \mathcal{L}(\mathbf{T})^c$. To show $\mathcal{C}l_{P,F,SA,SP,FT}(\mathbf{T}) \subseteq \{\mathbf{T}_{\mathcal{L}'} : \mathcal{L}' \preceq \mathcal{L}\}$, it is sufficient to show the stability of $\{\mathbf{T}_{\mathcal{L}'} : \mathcal{L}' \preceq \mathcal{L}\}$ by all the operations. Let $\mathcal{L}_1 \subseteq \mathcal{E}^{d_1}_{\mathcal{A}_1}$ and $\mathcal{L}_2 \subseteq \mathcal{E}^{d_2}_{\mathcal{A}_2}$ be two languages such that $\mathcal{L}_i \preceq \mathcal{L}$ for $i \in \{1, 2\}$. Thus, for $i \in \{1, 2\}$, there exists a Turing machine $\mathcal{M}_i$ with semi-oracle $\mathcal{L}$ whose domain is exactly $\mathcal{L}_i$.

• **Stability under product:** Let $\mathbf{T}' = \phi_P(\mathbf{T}_1, \mathbf{T}_2)$, so $\mathbf{T}' = \mathbf{T}_{\mathcal{L}'}$ with $\mathcal{L}' = \mathcal{L}_1 \times \mathcal{E}^{d_2}_{\mathcal{A}_2} \cup \mathcal{E}^{d_1}_{\mathcal{A}_1} \times \mathcal{L}_2$. The language $\mathcal{L}'$ could be the domain of a Turing machine $\mathcal{M}'$ with semi-oracle $\mathcal{L}$. It suffices to simulate the two Turing machines $\mathcal{M}_1$ and $\mathcal{M}_2$ (each machine runs during one step successively) on each coordinate of a pattern of $\mathcal{L}'$. Thus $\mathcal{L}' \preceq \mathcal{L}$.

• **Stability under finite type:** Let $\mathbf{T}' = \phi_{FT}(P, \mathbf{T}_{\mathcal{L}_1})$. Since P is finite, one has $\mathcal{L}_1 \cup P \preceq \mathcal{L}_1 \preceq \mathcal{L}$ and $\mathbf{T}' = \mathbf{T}_{\mathcal{L}_1 \cup P}$.

• **Stability under factor map:** Let $\mathbf{T}' = \phi(\pi, \mathbf{T}_{\mathcal{L}_1})$ where $\pi : \mathcal{A}_1^{\mathbb{Z}^{d_1}} \to \mathcal{B}^{\mathbb{Z}^{d_1}}$ is a morphism of neighborhood $\mathbb{S}^{d_1}_n$ and local function $\overline{\pi}$. One has $\mathbf{T}' = \mathbf{T}_{\mathcal{L}'}$ where $\mathcal{L}' = (\overline{\pi}(\mathcal{L}_1^c))^c$. Moreover, one has $\mathcal{L}' \preceq \mathcal{L}_1$. Indeed, if $p \in \mathcal{E}^{d_1}_{\mathcal{B}}$, we simulate the machine $\mathcal{M}_1$ on all pattern $p' \in \mathcal{A}^{supp(p)+\mathbb{S}^{d_1}_n}$ such that $\overline{\pi}(p') = p$, running successively one step for each pattern.

• **Stability under sub-action:** Let $\mathbf{T}' = \phi_{SA}(\mathbb{G}, \mathbf{T}_{\mathcal{L}_1}) \subseteq \mathcal{A}_1^{\mathbb{Z}^{d'}}$ where $\mathbb{G}$ is a subgroup of $\mathbb{Z}^{d_1}$ of dimension $d' \leq d_1$. We consider the language $\mathcal{L}' \subseteq \mathcal{E}^{d_1}_{\mathcal{A}_1}$ which is the domain of the Turing machine $\mathcal{M}'$: on a pattern $p \in \mathcal{E}^{d'}_{\mathcal{A}_1}$ of support $\mathbb{U}$, a Turing machine $\mathcal{M}'$ simulates successively $\mathcal{M}_1$ on every entry word of support $[-n; n]^{d_1}$ which completes p in $\mathcal{E}^{d_1}_{\mathcal{A}_1}$ where $[-n; n]^{d_1}$ is the minimal support which contains $\mathbb{U}$ embedded in $\mathbb{G}$. Thus $\mathcal{L}' \preceq \mathcal{L}_1$, moreover $\mathbf{T}' = \mathbf{T}_{\mathcal{L}'}$. This is exactly the same principle as in the proof of Theorem 2.2.

• **Stability under superposition:** Let $\mathbf{T}' = \phi_{SP}(\mathbb{G}, \mathbb{G}', \mathbf{T}_{\mathcal{L}_1})$ where $\mathbb{G}$ is isomorph to $\mathbb{Z}^{d_1}$ and $\mathbb{G} \oplus \mathbb{G}' = \mathbb{Z}^{d_1+d}$. Let $\mathcal{L}' \subseteq \mathcal{E}^{d_1+d}_{\mathcal{A}_1}$ be the language where each pattern p is the superposition of patterns $p_1, \dots, p_d \in \mathcal{E}^{d_1}_{\mathcal{A}_1}$ and there exists $i \in \{1, \dots, d\}$ such that $p_i \in \mathcal{L}_1$. Thus $\mathcal{L}' \preceq \mathcal{L}_1$ and $\mathbf{T}' = \mathbf{T}_{\mathcal{L}'}$.

4.2.2. *Reciprocal inclusion.* Let $\mathbf{T} \subseteq \mathcal{A}^{\mathbb{Z}^d}$ be a subshift; define $\mathcal{L} = \mathcal{L}(\mathbf{T})^c \subseteq \mathcal{E}_{\mathcal{A}}^d$. Let $\mathcal{L}' \subseteq \mathcal{E}_{\mathcal{B}}^{d'}$ be a language such that $\mathcal{L}' \preceq \mathcal{L}$. We want to prove that $\mathbf{T}_{\mathcal{L}'} \in \mathcal{C}l_{P,F,SA,SP,FT}(\mathbf{T})$.

Here, we assume that $\mathcal{L}$ and $\mathcal{L}'$ are one-dimensional languages, but the proof can be adapted to the general case. We explain how to construct the subshift $\mathbf{T}_{\mathcal{L}'}$ thanks to operations P, F, FT, SA and SP applied on $\mathbf{T} = \mathbf{T}_{\mathcal{L}}$.

Since $\mathcal{L}' \preceq \mathcal{L}$ there exists a Turing machine $\mathcal{M}$ with semi-oracle $\mathcal{L}$ such that $dom(\mathcal{M}) = \mathcal{L}'$. We transform this Turing machine so that it only takes in input patterns of support $[0, 2^{n-1}]$ (because checked patterns are given by $\mathbf{W}_2$) and at the moment when the state $q_?$ is reached, the word written on the oracle tape is copied out in the alphabet $\widetilde{\mathcal{A}}$, which is simply a copy of $\mathcal{A}$, then again copied out in the alphabet $\mathcal{A}$ once the oracle has given its answer.

We first list auxiliary subshifts that we need to construct $\mathbf{T}_{\mathcal{L}'}$:

- the original subshift $\mathbf{T}_{\mathcal{L}}$ written in the copy of $\mathcal{A}$: $\widetilde{\mathbf{T}}_{\mathcal{L}} \subseteq \widetilde{\mathcal{A}}^{\mathbb{Z}}$ will simulate the oracle;
- Turing machine $\mathcal{M}$ is coded by a subshift of finite type $\mathbf{T}_{\mathcal{M}} \subseteq \mathcal{O}^{\mathbb{Z}^2}$, where $\mathcal{O}$ is an alphabet that contains at least $\mathcal{A}$, $\widetilde{\mathcal{A}}$ and $\mathcal{B}$;
- the framework for this Turing machine will be given by $\mathbf{W}_2$, $\mathbf{W}_3$ and $\mathbf{W}_5$ defined in Section 3; they are defined on the alphabet $\{\bullet, \circ\}$ and are subshifts of finite type up to a letter-to-letter morphism.

Construction of $\mathbf{T}_{\mathcal{L}'}$. The principle is to construct $\mathbf{\Sigma} \in \mathcal{C}l_{P,F,SA,SP,FT}(\mathbf{T}_{\mathcal{L}})$ a 4-dimensional subshift on the alphabet $\mathcal{C} = \mathcal{A} \times \widetilde{\mathcal{A}} \times \mathcal{B} \times \{\bullet; \circ\}^3 \times \mathcal{O}$. Denote (e_1, e_2, e_3, e_4) the canonical basis of $\mathbb{Z}^4$. We need these four dimensions for different reasons :

- the subshift $\mathbf{T}_{\mathcal{L}'}$ will appear on $\mathbb{Z}e_1$;
- thanks to $\mathbb{Z}e_1 \oplus \mathbb{Z}e_2 \oplus \mathbb{Z}e_3$, we construct a framework for $\mathcal{M}$, so that every rectangle of this framework is in a plane $\{i\} \times \mathbb{Z} \times \mathbb{Z} \times \{k\}$ where $i, k \in \mathbb{Z}$;
- on $\mathbb{Z}e_4$ we have the oracle simulated by $\widetilde{\mathbf{T}}_{\mathcal{L}}$.

Step 1 : First notice that changing $\mathbf{T}_{\mathcal{L}}$ into $\widetilde{\mathbf{T}}_{\mathcal{L}}$ only requires a letter-to-letter morphism. Then we construct $\tilde{\mathbf{W}} = \phi_{SP}(\mathbb{Z}e_4, \mathbb{Z}e_1 \oplus \mathbb{Z}e_2 \oplus \mathbb{Z}e_3, \widetilde{\mathbf{T}}_{\mathcal{L}})$ to place $\widetilde{\mathbf{T}}_{\mathcal{L}}$ in a 4-dimensional subshift. We finally add through a product operation P all letters from $\mathcal{C}$: $\mathbf{W} = \tilde{\mathbf{W}} \times (\mathcal{A} \times \mathcal{B} \times \{\bullet; \circ\}^3 \times \mathcal{O})^{\mathbb{Z}^4}$ so that $\mathbf{W} \in \mathcal{C}l_{P,F,SP}(\mathbf{T}_{\mathcal{L}}) \cap \mathcal{C}^{\mathbb{Z}^4}$.

Step 2 : We want $\mathbf{T}_{\mathcal{L}'}$ to appear on $\mathbb{Z}e_1$. Simulations of the Turing machine $\mathcal{M}$ will take in input a word written on $\mathbb{Z}e_2$. So we need to copy out $\mathbb{Z}e_1$ on $\mathbb{Z}e_2$ so that these simulations apply to what will be the subshift $X_{\mathcal{L}'}$. We get to it with the finite condition :

$$\forall x \in \mathcal{C}^{\mathbb{Z}^4}, \forall u \in \mathbb{Z}^4, x_u = x_{u+e_1-e_2}.$$

We also want to keep accessible all along the simulation the entry word of every rectangle of the framework. To do that we add the finite condition :

$$\forall x \in \mathcal{C}^{\mathbb{Z}^4}, \forall u \in \mathbb{Z}^4, x_u = x_{u+e_3}.$$

We thus obtain a subshift $\mathbf{W}' \in \mathcal{C}l_{P,F,SP,FT}(\mathbf{T}_{\mathcal{L}})$.

Step 3 : Then we add to $\mathbf{W}'$ a framework for the Turing machine. We construct $W_{\text{rect}} \subseteq \{\bullet, \circ\}^{\mathbb{Z}^3}$ an auxiliary subshift of finite type up to a letter-to-letter morphism, containing well-chosen rectangles. Denote F_i the finite type condition that ensures $\forall x \in \{\bullet, \circ\}^{\mathbb{Z}^3}, \forall u \in \mathbb{Z}^3, x_u = x_{u+e_i}$. As in Section 3, we define:

- $\mathbf{W}_2 = \phi_{FT}(F_3, \phi_{SP}(\mathbb{Z}e_1 \oplus \mathbb{Z}e_2, \mathbb{Z}e_3 \oplus \mathbb{Z}e_4, \mathbf{S}_2^{(\uparrow)}))$;
- $\mathbf{W}_5 = \phi_{FT}(F_3, \phi_{SP}(\mathbb{Z}e_1 \oplus \mathbb{Z}e_2, \mathbb{Z}e_3 \oplus \mathbb{Z}e_4, \mathbf{S}_5^{(\uparrow)}))$;
- $\mathbf{W}_3 = \phi_{FT}(F_2, \phi_{SP}(\mathbb{Z}e_1 \oplus \mathbb{Z}e_3, \mathbb{Z}e_2 \oplus \mathbb{Z}e_4, \mathbf{S}_3^{(\uparrow)}))$.

The rectangles are obtained in $\tilde{\mathbf{W}}_{\text{rect}} = \mathbf{W}_2 \times \mathbf{W}_5 \times \mathbf{W}_3$. Each rectangle of length 5^m given by $\mathbf{W}_5$ knows the length of its input 2^n given by $\mathbf{W}_2$. Thus we can simulate the Turing machine on words of length 2^n, on a tape of length 5^m and simulations are bounded by 3^p steps of calculation. Up to a letter-to-letter morphism, $\tilde{\mathbf{W}}_{\text{rect}}$ is a subshift of finite type, so there exists a finite set of patterns F_{rect} and a morphism π_{rect} such that $\tilde{\mathbf{W}}_{\text{rect}} = \pi_{\text{rect}}(\mathbf{T}_{F_{\text{rect}}})$. We add this framework to $\mathbf{W}'$ via $\mathbf{W}_{\text{rect}} = \pi_{\text{rect}}(\phi_{FT}(F_{\text{rect}}, \mathbf{W}'))$ so that we have $\mathbf{W}_{\text{rect}} \in \mathcal{C}l_{P,F,SP,FT}(\mathbf{T}_{\mathcal{L}})$.

Step 4 : We add the behaviour of $\mathcal{M}$ in rectangles of $\mathbf{W}_{\text{rect}}$ but for the moment we do not take into consideration calls for oracle. As in Section 3, we consider the finite conditions $P_{\mathcal{M}}$ given by the rule of $\mathcal{M}$ and the conditions $P_{\text{calc}} = \{\mathbf{Init}, \mathbf{Head}, \mathbf{Stop}, \mathbf{Final}\}$ which control the interaction of the head of $\mathcal{M}$ with the rectangles. For the moment every time the machine calls the oracle it keeps on calculating. Thus $\mathbf{W}_{\mathcal{M}} = \phi_{FT}(P_{\mathcal{M}} \cup P_{\text{calc}}, \mathbf{W}_{\text{rect}}) \in \mathcal{C}l_{P,F,FT,SP}(\mathbf{T}_{\mathcal{L}})$.

Step 5 : To simulate the oracle, we add finite type conditions to ensure that during a calculation, when the machine calls for the oracle in $(i, j, k, l) \in \mathbb{Z}^4$, the pattern $p \in \tilde{\mathcal{A}}^n$ on which the oracle is called coincides with the pattern in $\mathbb{Z}e_4$ between (i, j, k, l) and $(i, j, k, l + n)$. These new allowed patterns look like :

$\uparrow_{e_4}$ $\begin{array}{|c|c|}\hline \tilde{a} & . \\ \hline \tilde{b} & \tilde{a} \\ \hline \end{array}$, $\begin{array}{|c|c|}\hline \tilde{a} & . \\ \hline (q_?, \tilde{b}) & \tilde{a} \\ \hline \end{array}$

$\rightarrow_{e_2}$

However, these conditions are only valid in the interior of a rectangle. We denote these finite type conditions by F_{oracle}. Then we have $\mathbf{W}_{\mathcal{M}_{\text{oracle}}} = \phi_{FT}(F_{\text{oracle}}, W_{\mathcal{M}}) \in \mathcal{C}l_{P,F,SP,FT}(\mathbf{T}_{\mathcal{L}})$.

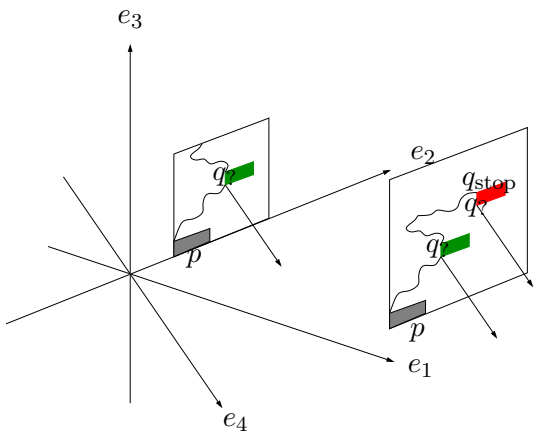
Step 6 : In order to avoid dependence problems between different calculations, each configuration of $\mathbf{T}_{\mathcal{L}}$ that appears on $\mathbb{Z}^4$ is used for the same calculation, thanks to the finite type condition :

$$\forall x \in \mathcal{C}^{\mathbb{Z}^4}, \forall u \in \mathbb{Z}^4, x_u = x_{u+e_1+e_4}.$$

Finally we consider the final state q_{stop} as a forbidden pattern and we denote by $\mathbf{\Sigma}$ this subshift. We have $\mathbf{\Sigma} \in \mathcal{C}l_{P,F,SP,FT}(\mathbf{T}_{\mathcal{L}})$.

We simulate the running of the Turing machine $\mathcal{M}$ on a pattern $p \in \mathcal{E}^1_{\mathcal{B}}$ of length 2^n. As soon as $\mathcal{M}$ calls for the oracle, we compare the word on which the oracle is called and the word on $\mathbb{Z}e_4$. If the two words coincide then $\mathcal{M}$ keeps on calculating, else it comes to the final state q_{stop}. If the machine cannot terminate its calculation within the time given by the rectangle, Proposition 3.4 ensures that we can find a larger rectangle in which the machine will calculate on the same entry word.

The following picture resumes the behaviour of the machine $\mathcal{M}$ on the framework :



Proof that this construction works. We now prove that $\phi_{SA}(\mathbb{Z}e_1, \mathbf{\Sigma})$, the projection of $\mathbf{\Sigma}$ on $\mathbb{Z}e_1$ is $\mathbf{T}_{\mathcal{L}'}$, up to a morphism that just consists in keeping information about $\mathcal{B}$.

Proof of $\phi_{SA}(\mathbb{Z}e_1, \mathbf{\Sigma}) \subseteq \mathbf{T}_{\mathcal{L}'}$: Let $y \in \mathbf{\Sigma}$, we prove that $x = y_{|\mathbb{Z}e_1} \in \mathbf{T}_{\mathcal{L}'}$. It is sufficient to prove that every pattern in x is not in $\mathcal{L}'$. Let p be a pattern in x; it is a sub-pattern of a certain $p' \sqsubset x$ where p' is chosen such that it is of length 2^n. By construction of $\mathbf{W}_{\text{rect}}$ there exists $t, s \in \mathbb{N}$ arbitrary large such that there exists a rectangle of size $5^s \times 3^t$ with the entry word p'. Since $y \in \mathbf{\Sigma}$, in every rectangle the calculation of the machine $\mathcal{M}$ on the word p' does not reach the final state q_{stop}. Since these rectangles are arbitrarily large, we can conclude that the machine $\mathcal{M}$ never reaches q_{stop}. It means that $p' \notin \mathcal{L}'$, thus $p \notin \mathcal{L}'$.

Proof of $\mathbf{T}_{\mathcal{L}'} \subseteq \phi_{SA}(\mathbb{Z}e_1, \mathbf{\Sigma})$: Let $x \in \mathbf{T}_{\mathcal{L}'}$, we construct $y \in \mathcal{C}^{\mathbb{Z}^4}$ such that $y \in \mathbf{\Sigma}$ and $y_{|\mathbb{Z}e_1} = x$. To insure that $y \in \mathbf{\Sigma}$ we just need to check that for all $(i, j, k) \in \mathbb{Z}^3$, we can impose that $y_{|\{i\}\times\{j\}\times\{k\}\times\mathbb{Z}} \in \mathbf{T}_{\mathcal{L}}$ while the calculations of $\mathcal{M}$ in the rectangles containing any (i, j, k, l) do not reach the state q_{stop}.

Let us now focus on a specific rectangle of the framework, on which the machine $\mathcal{M}$ calculates on a pattern p of size 2^n that appears in x. Since p appears in x, $p \notin \mathcal{L}'$ so the machine $\mathcal{M}$ loops on the entry p. It means that every time the calculation of $\mathcal{M}$ on p calls for the oracle on a pattern p', p' is not in $\mathcal{L}$. Since $\mathcal{L} = \mathcal{L}(\mathbf{T})^c$, for all pattern p' on which the oracle is called, there exists a configuration $z \in \mathbf{T}_{\mathcal{L}}$ such that $z_{|[0;|m'|-1]} = p'$. Thus we complete y on the following way :

- if in $(i, j, k) \in \mathbb{Z}^3$ the calculation of $\mathcal{M}$ calls for the oracle on a pattern p', then $y_{|\{i\}\times\{j\}\times\{k\}\times\mathbb{Z}} = z$ previously constructed;
- if the oracle is not called, we complete y with any $y_{|\{i\}\times\{j\}\times\{k\}\times\mathbb{Z}} \in \mathbf{T}_{\mathcal{L}}$.

This makes sure that y is in the subshift $\mathbf{\Sigma}$, so $x \in \phi_{SA}(\mathbb{Z}e_1, \mathbf{\Sigma})$.

The proof of Theorem is completed. ∎

An application of Theorem 4.2: There does not exist an "universal" subshift $\mathbf{T}$ which could simulate every element of $\mathcal{S}$. Indeed, consider $\mathcal{L} = \mathcal{L}(\mathbf{T})^c$, one has $\mathcal{C}l_{P,F,SA,SP,FT}(\mathbf{T}) = \{\mathbf{T}_{\mathcal{L}'} : \mathcal{L}' \preceq \mathcal{L}\}$. But there exists $\mathcal{L}''$ strictly superior to $\mathcal{L}$ (see [RJ87]). Moreover, one can choose $\mathcal{L}''$ such that for all patterns $p \in \mathcal{L}'' \subseteq \mathcal{E}^d_{\mathcal{A}}$, then for all $p' \in \mathcal{E}^d_{\mathcal{A}}$ such that $p \sqsubset p'$, one has $p' \in \mathcal{L}''$. Thus $\mathcal{L}(\mathbf{T}_{\mathcal{L}''})^c = \mathcal{L}''$. One deduces that $\mathbf{T}_{\mathcal{L}''} \notin \mathcal{C}l_{P,F,SA,SP,FT}(\mathbf{T})$.

Conclusion

In this article we generalize the notion of tilings considering any set of forbidden patterns. We present operations on sets of tilings, called subshifts, inspired by the dynamical theory. We obtain different notions of simulation, depending on the set of operations which are considered. These notions involve different semi-orders on subshifts and in this article we focus on the semi-order which consider all the transformations presented. This semi-order is quite well understood since we establish a correspondence with a semi-order on languages of forbidden patterns based on computability properties. The following points are still open questions :

- In our construction, considering two subshifts $\mathbf{T}_1$ and $\mathbf{T}_2$ respectively of dimension d_1 and d_2 such that $\mathcal{L}(\mathbf{T}_2)^c \preceq \mathcal{L}(\mathbf{T}_1)^c$, we need $\mathbf{\Sigma} \in \mathcal{C}l_{P,F,SA,SP,FT}(\mathbf{T}_1)$ of dimension $d_1 + d_2 + 2$ to simulate $\mathbf{T}_2$. It is possible to decrease the dimension of $\mathbf{\Sigma}$?
- For which class $\mathcal{U} \subseteq \mathcal{S}$ there exists a subshift $\mathbf{T}$ such that $\mathcal{C}l_{P,F,SA,SP,FT}(\mathbf{T}) = \mathcal{U}$?

We can also consider other semi-orders involved by other sets of operations and look for general tools to study them. In fact, some of these semi-orders have already been studied. For example, the set of space-time diagrams of a cellular automaton can be viewed as a subshift, and the orders presented in [MR99, Oll03, The05] could be formalized with the tools introduced in Section 2.

References

[BDJ08] Alexis Ballier, Bruno Durand, and Emmanuel Jeandel. Structural Aspects of Tilings. In *Proceedings of the 25th Symposium on Theoretical Aspects of Computer Science : STACS 2008*, 2008.

[Bea93] M.P. Beal. *Codage Symbolique*. Masson, 1993.

[Ber66] R. Berger. *The Undecidability of the Domino Problem*. American Mathematical Society, 1966.

[Han74] William Hanf. Nonrecursive tilings of the plane. i. *The Journal of Symbolic Logic*, 39(2):283–285, 1974.

[Hed69] GA Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Theory of Computing Systems*, 3(4):320–375, 1969.

[Hoc07] M. Hochman. On the Dynamics and Recursive Properties of Multidimensional Symbolic Systems. 2007.

[Kit98] B. Kitchens. *Symbolic dynamics*. Springer New York, 1998.

[LM95] D. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

[Moz89] S. Mozes. Tilings, substitution systems and dynamical systems generated by them. *Journal d'analyse mathématique(Jerusalem)*, 53:139–186, 1989.

[MR99] J. Mazoyer and I. Rapaport. Inducing an order on cellular automata by a grouping operation. *Discrete Applied Mathematics*, 91(1-3):177–196, 1999.

[Oll03] N. Ollinger. The intrinsic universality problem of one-dimensional cellular automata. *Symposium on Theoretical Aspects of Computer Science (STACS'2003)*, LNCS:632–641, 2003.

[RJ87] H. Rogers Jr. *Theory of recursive functions and effective computability*. MIT Press Cambridge, MA, USA, 1987.

[Rob71] R.M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12(3):177–209, 1971.

[The05] G. Theyssier. *Automates cellulaires: un modèle de complexités*. PhD thesis, École Normale Supérieure de Lyon, 2005.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 111–122
www.stacs-conf.org

COMPRESSED REPRESENTATIONS OF PERMUTATIONS, AND APPLICATIONS

JÉRÉMY BARBAY AND GONZALO NAVARRO

Dept. of Computer Science (DCC), University of Chile.
E-mail address: {jbarbay,gnavarro}@dcc.uchile.cl

Abstract. We explore various techniques to compress a permutation π over n integers, taking advantage of ordered subsequences in π, while supporting its application $\pi(i)$ and the application of its inverse $\pi^{-1}(i)$ in small time. Our compression schemes yield several interesting byproducts, in many cases matching, improving or extending the best existing results on applications such as the encoding of a permutation in order to support iterated applications $\pi^k(i)$ of it, of integer functions, and of inverted lists and suffix arrays.

1. Introduction

Permutations of the integers $[n] = \{1, \ldots, n\}$ are a basic building block for the succinct encoding of integer functions [38], strings [1, 18, 39, 41], and binary relations [5, 4], among others. A permutation π is trivially representable in $n\lceil \lg n \rceil$ bits, which is within $\mathcal{O}(n)$ bits of the information theory lower bound of $\lg(n!)$ bits.[1] In many interesting applications, efficient computation of both the permutation $\pi(i)$ and its inverse $\pi^{-1}(i)$ is required.

The lower bound of $\lg(n!)$ bits yields a lower bound of $\Omega(n \log n)$ comparisons to sort such a permutation in the comparison model. Yet, a large body of research has been dedicated to finding better sorting algorithms which can take advantage of specificities of each permutation to sort. Trivial examples are permutations sorted such as the identity, or containing sorted blocks [32] (e.g. $(1, 3, 5, 7, 9, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}, \mathbf{10})$ or $(6, 7, 8, 9, 10, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5})$), or containing sorted subsequences [28] (e.g. $(1, \mathbf{6}, 2, \mathbf{7}, 3, \mathbf{8}, 4, \mathbf{9}, 5, \mathbf{10})$): algorithms performing only $\mathcal{O}(n)$ comparisons on such permutations, yet still $\mathcal{O}(n \log n)$ comparisons in the worst case, are achievable and obviously preferable. Less trivial examples are classes of permutations whose structure makes them interesting for applications: see Mannila's seminal paper [32] and Estivil-Castro and Wood's review [14] for more details.

Each sorting algorithm in the comparison model yields an encoding scheme for permutations: It suffices to note the result of each comparison performed to uniquely identify the permutation sorted, and hence to encode it. Since an adaptive sorting algorithm performs $o(n \log n)$ comparisons on many classes of permutations, each adaptive algorithm yields a *compression scheme* for permutations, at the cost of losing a constant factor on some other

Key words and phrases: Compression, Permutations, Succinct Data Structures, Adaptive Sorting.
Second author partially funded by Fondecyt Grant 1-080019, Chile.

[1] In this paper we use the notations $\lg x = \log_2 x$ and $[x] = \{1, \ldots, x\}$.

"bad" classes of permutations. We show in Section 4 some examples of applications where only "easy" permutations arise. Yet such compression schemes do not necessarily support in reasonable time the inverse of the permutation, or even the simple application of the permutation: this is the topic of our study. We describe several encodings of permutations so that on interesting classes of instances the encoding uses $o(n \log n)$ bits while supporting the operations $\pi(i)$ and $\pi^{-1}(i)$ in time $o(\log n)$. Later, we apply our compression schemes to various scenarios, such as the encoding of integer functions, text indexes, and others, yielding original compression schemes for these abstract data types.

2. Previous Work

Definition 2.1. The *entropy* of a sequence of positive integers $X = \langle n_1, n_2, \ldots, n_r \rangle$ adding up to n is $H(X) = \sum_{i=1}^{r} \frac{n_i}{n} \lg \frac{n}{n_i}$. By convexity of the logarithm, $\frac{r \lg n}{n} \leq H(X) \leq \lg r$.

Succinct Encodings of Sequences. Let $S[1, n]$ be a sequence over an alphabet $[r]$. This includes bitmaps when $r = 2$ (where, for convenience, the alphabet will be $\{0, 1\}$). We will make use of succinct representations of S that support operations *rank* and *select*: $rank_c(S, i)$ gives the number of occurrences of c in $S[1, i]$ and $select_c(S, j)$ gives the position in S of the jth occurrence of c.

For the case $r = 2$, S requires n bits of space and *rank* and *select* can be supported in constant time using $\mathcal{O}(\frac{n \log\log n}{\log n}) = o(n)$ bits on top of S [36, 10, 17]. The extra space is more precisely $\mathcal{O}(\frac{n \log b}{b} + 2^b \operatorname{polylog}(b))$ for some parameter b, which is chosen to be, say, $b = \frac{1}{2} \lg n$ to achieve the given bounds. In this paper, we will sometimes apply the technique over sequences of length $\ell = o(n)$ (n will be the length of the permutations). Still, we will maintain the value of b as a function of n, not ℓ, which ensures that the extra space will be of the form $\mathcal{O}(\frac{\ell \log\log n}{\log n})$, i.e., it will tend to zero when divided by ℓ as n grows, even if ℓ stays constant. All of our $o()$ terms involving several variables in this paper can be interpreted in this strong sense: asymptotic in n. Thus we will write the above space simply as $o(\ell)$.

Raman *et al.* [40] devised a bitmap representation that takes $nH_0(S) + o(n)$ bits, while maintaining the constant time for the operations. Here $H_0(S) = H(\langle n_1, n_2, \ldots, n_r \rangle) \leq \lg r$, where n_c is the number of occurrences of symbol c in S, is the so-called *zero-order entropy* of S. For the binary case this simplifies to $nH_0(S) = m \lg \frac{n}{m} + (n-m) \lg \frac{n}{n-m} = m \lg \frac{n}{m} + \mathcal{O}(m)$, where m is the number of bits set in S.

Grossi *et al.* [19] extended the result to larger alphabets using the so-called *wavelet tree*, which decomposes a sequence into several bitmaps. By representing those bitmaps in plain form, one can represent S using $n\lceil \lg r \rceil (1 + o(1))$ bits of space, and answer $S[i]$, as well as *rank* and *select* queries on S, in time $\mathcal{O}(\log r)$. By, instead, using Raman *et al.*'s representation for the bitmaps, one achieves $nH_0(S) + o(n \log r)$ bits of space, and the same times. Ferragina *et al.* [15] used multiary wavelet trees to maintain the same compressed space, while improving the times for all the operations to $\mathcal{O}(1 + \frac{\log r}{\log\log n})$.

Measures of Disorder in Permutations. Various previous studies on the presortedness in sorting considered in particular the following measures of order on an input array to be sorted. Among others, Mehlhorn [34] and Guibas *et al.* [21] considered the number of pairs in the wrong order, Knuth [27] considered the number of ascending substrings (runs), Cook and Kim [12], and later Mannila [32] considered the number of elements which have to be

removed to leave a sorted list, Mannila [32] considered the smallest number of exchanges of arbitrary elements needed to bring the input into ascending order, Skiena [44] considered the number of encroaching sequences, obtained by distributing the input elements into sorted sequences built by additions to both ends, and Levcopoulos and Petersson [28] considered Shuffled UpSequences and Shuffled Monotone Sequences. Estivil-Castro and Wood [14] list them all and some others.

3. Compression Techniques

We first introduce a compression method that takes advantage of (ascending) runs in the permutation. Then we consider a stricter variant of the runs, which allows for further compression in applications when those runs arise, and in particular allows the representation size to be sublinear in n. Next, we consider a more general type of runs, which need not be contiguous.

3.1. Wavelet Tree on Runs

One of the best known sorting algorithm is merge sort, based on a simple linear procedure to merge two already sorted arrays, resulting in a worst case complexity of $\mathcal{O}(n \log n)$. Yet, checking in linear time for *down-step* positions in the array, where an element is followed by a smaller one, partitions the original arrays into ascending runs which are already sorted. This can speed up the algorithm when the array is partially sorted [27]. We use this same observation to encode permutations.

Definition 3.1. A *down step* of a permutation π over $[n]$ is a position i such that $\pi(i+1) < \pi(i)$. A *run* in a permutation π is a maximal range of consecutive positions $\{i, \ldots, j\}$ which does not contain any down step. Let $d_1, d_2, \ldots, d_k$ be the list of consecutive down steps in π. Then the number of runs of π is noted $\rho = k + 1$, and the sequence of the lengths of the runs is noted $\texttt{Runs} = \langle d_1, d_2 - d_1, \ldots, d_k - d_{k-1}, n + 1 - d_k \rangle$.

For example, permutation $(1, 3, 5, 7, 9, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}, \mathbf{10})$ contains $\rho = 2$ runs, of lengths $\langle 5, 5 \rangle$. Whereas previous analyses [32] of adaptive sorting algorithms considered only the number ρ of runs, we refine them to consider the distribution Runs of the sizes of the runs.

Theorem 3.2. *There is an encoding scheme using at most* $n(2 + H(\texttt{Runs}))(1 + o(1)) + \mathcal{O}(\rho \log n)$ *bits to encode a permutation* π *over* $[n]$ *covered by* ρ *runs of lengths* Runs. *It supports* $\pi(i)$ *and* $\pi^{-1}(i)$ *in time* $\mathcal{O}(1 + \log \rho)$ *for any value of* $i \in [n]$. *If* i *is chosen uniformly at random in* $[n]$ *then the average time is* $\mathcal{O}(1 + H(\texttt{Runs}))$.

Proof. The Hu-Tucker algorithm [23] (see also Knuth [27, p. 446]) produces in $\mathcal{O}(\rho \log \rho)$ time a prefix-free code from a sequence of frequencies $X = \langle n_1, n_2, \ldots, n_\rho \rangle$ adding up to n, so that (1) the i-th lexicographically smallest code is that for frequency n_i, and (2) if ℓ_i is the bit length of the code assigned to the i-th sequence element, then $L = \sum \ell_i n_i$ is minimal and moreover $L < n(2 + H(X))$ [27, p. 446, Eq. (27)].

We first determine Runs in $\mathcal{O}(n)$ time, and then apply the Hu-Tucker algorithm to Runs. We arrange the set of codes produced in a binary trie (equivalent to a Huffman tree [24]), where each leaf corresponds to a run and points to its two endpoints in π. Because of property (1), reading the leaves left-to-right yields the runs also in left-to-right order. Now we convert this trie into a wavelet-tree-like structure [19] without altering its shape,

as follows. Starting from the root, first process recursively each child. For the leaves do nothing. Once both children of an internal node have been processed, the invariant is that they point to the contiguous area in π covering all their leaves, and that this area of π has already been sorted. Now we merge the areas of the two children in time proportional to the new area created (which, again, is contiguous in π because of property (1)). As we do the merging, each time we take an element from the left child we append a 0 bit to a bitmap we create for the node, and a 1 bit when we take an element from the right list.

When we finish, we have the following facts: (1) π has been sorted, (2) the time for sorting has been $\mathcal{O}(n + \rho \log \rho)$ plus the total number of bits appended to all bitmaps, (3) each of the n_i elements of leaf i (at depth ℓ_i) has been merged ℓ_i times, contributing ℓ_i bits to the bitmaps of its ancestors, and thus the total number of bits is $\sum n_i \ell_i$.

Therefore, the total number of bits in the Hu-Tucker-shaped wavelet tree is at most $n(2+H(\texttt{Runs}))$. To this we must add the $\mathcal{O}(\rho \log n)$ bits of the tree pointers. We preprocess all the bitmaps for *rank* and *select* queries so as to spend $o(n(2+H(\texttt{Runs}))$ extra bits (§2).

To compute $\pi^{-1}(i)$ we start at offset i at the root bitmap B, with position $p \leftarrow 0$, and bitmap size $s \leftarrow n$. If $B[i] = 0$ we go down to the left child with $i \leftarrow rank_0(B, i)$ and $s \leftarrow rank_0(B, s)$. Otherwise we go down to the right child with $i \leftarrow rank_1(B, i)$, $p \leftarrow p + rank_0(B, s)$, and $s \leftarrow rank_1(B, s)$. When we reach a leaf, the answer is $p + i$.

To compute $\pi(i)$ we do the reverse process, but we must first determine the leaf v and offset j within v corresponding to position i: We start at the root bitmap B, with bitmap size $s \leftarrow n$ and position $j \leftarrow i$. If $rank_0(B, s) \geq j$ we go down to the left child with $s \leftarrow rank_0(B, s)$. Otherwise we go down to the right child with $j \leftarrow j - rank_0(B, s)$ and $s \leftarrow rank_1(B, s)$. We eventually reach leaf v, and the offset within v is j. We now start an upward traversal using the nodes that are already in the recursion stack (those will be limited to $\mathcal{O}(\log \rho)$ soon). If v is a left child of its parent u, then we set $j \leftarrow select_0(B, j)$, else we set $j \leftarrow select_1(B, j)$, where B is the bitmap of u. Then we set $v \leftarrow u$ until reaching the root, where $j = \pi(i)$.

In both cases the time is $\mathcal{O}(\ell)$, where ℓ is the depth of the leaf arrived at. If i is chosen uniformly at random in $[n]$, then the average cost is $\frac{1}{n} \sum n_i \ell_i = \mathcal{O}(1 + H(\texttt{Runs}))$. However, the worst case can be $\mathcal{O}(\rho)$ in a fully skewed tree. We can ensure $\ell = \mathcal{O}(\log \rho)$ in the worst case while maintaining the average case by slightly rebalancing the Hu-Tucker tree: If there exist nodes at depth $\ell = 4 \lg \rho$, we rebalance their subtrees, so as to guarantee maximum depth $5 \lg \rho$. This affects only marginally the size of the structure. A node at depth ℓ cannot add up to a frequency higher than $n/2^{\lfloor \ell/2 \rfloor} \leq 2n/\rho^2$ (see next paragraph). Added over all the possible ρ nodes we have a total frequency of $2n/\rho$. Therefore, by rebalancing those subtrees we add at most $\frac{2n \lg \rho}{\rho}$ bits. This is $o(n)$ if $\rho = \omega(1)$, and otherwise the cost was $\mathcal{O}(\rho) = \mathcal{O}(1)$ anyway. For the same reasons the average time stays $\mathcal{O}(1 + H(\texttt{Runs}))$ as it increases at most by $\mathcal{O}(\frac{\log \rho}{\rho}) = \mathcal{O}(1)$.

The bound on the frequency at depth ℓ is proved as follows. Consider the node v at depth ℓ, and its grandparent u. Then the uncle of v cannot have smaller frequency than v. Otherwise we could improve the already optimal Hu-Tucker tree by executing either a single (if v is left-left or right-right grandchild of u) or double (if v is left-right or right-left grandchild of u) AVL-like rotation that decreases the depth of v by 1 and increases that of the uncle of v by 1. Thus the overall frequency at least doubles whenever we go up two nodes from v, and this holds recursively. Thus the weight of v is at most $n/2^{\lfloor \ell/2 \rfloor}$. ∎

The general result of the theorem can be simplified when the distribution Runs is not particularly favorable.

Corollary 3.3. *There is an encoding scheme using at most* $n\lceil \lg \rho \rceil (1 + o(1)) + \mathcal{O}(\log n)$ *bits to encode a permutation* π *over* $[n]$ *with a set of* ρ *runs. It supports* $\pi(i)$ *and* $\pi^{-1}(i)$ *in time* $\mathcal{O}(1 + \log \rho)$ *for any value of* $i \in [n]$.

As a corollary, we obtain a new proof of a well-known result on adaptive algorithms telling that one can sort in time $O(n(1 + \log \rho))$ [32], now refined to consider the entropy of the partition and not only its size.

Corollary 3.4. *We can sort an array of length* n *covered by* ρ *runs of lengths* Runs *in time* $\mathcal{O}(n(1 + H(\texttt{Runs})))$, *which is worst-case optimal in the comparison model among all permutations with* ρ *runs of lengths* Runs *so that* $\rho \log n = o(nH(\texttt{Runs}))$.

3.2. Stricter Runs

Some classes of permutations can be covered by a small number of runs of a stricter type. We present an encoding scheme which uses $o(n)$ bits for encoding the permutations from those classes, and still $\mathcal{O}(n \lg n)$ bits for all others.

Definition 3.5. A *strict run* in a permutation π is a maximal range of positions satisfying $\pi(i + k) = \pi(i) + k$. The *head* of such run is its first position. The number of strict runs of π is noted τ, and the sequence of the lengths of the strict runs is noted SRuns. We will call HRuns the sequence of run lengths of the sequence formed by the strict run heads of π.

For example, permutation $(6, 7, 8, 9, 10, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5})$ contains $\tau = 2$ strict runs, of lengths $\texttt{SRuns} = \langle 5, 5 \rangle$. The run heads are $\langle 6, \mathbf{1} \rangle$, and contain 2 runs, of lengths $\texttt{HRuns} = \langle 1, 1 \rangle$. Instead, $(1, 3, 5, 7, 9, \mathbf{2}, \mathbf{4}, \mathbf{6}, \mathbf{8}, \mathbf{10})$ contains $\tau = 10$ strict runs, all of length 1.

Theorem 3.6. *There is an encoding scheme using at most* $\tau H(\texttt{HRuns})(1+o(1))+2\tau \lg \frac{n}{\tau} + o(n)+\mathcal{O}(\tau+\rho \log \tau)$ *bits to encode a permutation* π *over* $[n]$ *covered by* τ *strict runs and by* $\rho \leq \tau$ *runs, and with* HRuns *being the* ρ *run lengths in the permutation of strict run heads. It supports* $\pi(i)$ *and* $\pi^{-1}(i)$ *in time* $\mathcal{O}(1 + \log \rho)$ *for any value of* $i \in [n]$. *If* i *is chosen uniformly at random in* $[n]$ *then the average time is* $\mathcal{O}(1 + H(\texttt{HRuns}))$.

Proof. We first set up a bitmap R marking with a 1 bit the beginning of the strict runs. Set up a second bitmap R^{inv} such that $R^{inv}[i] = R[\pi^{-1}(i)]$. Now we create a new permutation π' of $[\tau]$ which collapses the strict runs of π, $\pi'(i) = rank_1(R^{inv}, \pi(select_1(R, i)))$. All this takes $\mathcal{O}(n)$ time and the bitmaps take $2\tau \lg \frac{n}{\tau} + \mathcal{O}(\tau) + o(n)$ bits using Raman *et al.*'s technique, where *rank* and *select* are solved in constant time (§2).

Now build the structure of Thm. 3.2 for π'. The number of down steps in π is the same as for the sequence of strict run heads in π, and in turn the same as the down steps in π'. So the number of runs in π' is also ρ and their lengths are HRuns. Thus we get at most $\tau(2+H(\texttt{HRuns}))(1+o(1))+\mathcal{O}(\rho \log \tau)$ bits to encode π', and can compute π' and its inverse in $\mathcal{O}(1 + \log \rho)$ worst case and $\mathcal{O}(1 + H(\texttt{HRuns}))$ average time.

To compute $\pi(i)$, we find $i' \leftarrow rank_1(R, i)$ and then compute $j' \leftarrow \pi'(i')$. The final answer is $select_1(R^{inv}, j')+i-select_1(R, i')$. To compute $\pi^{-1}(i)$, we find $i' \leftarrow rank_1(R^{inv}, i)$ and then compute $j' \leftarrow (\pi')^{-1}(i')$. The final answer is $select_1(R, j') + i - select_1(R^{inv}, i')$. This adds only constant time on top of that to compute π' and its inverse. ∎

Once again, we might simplify the results when the distribution HRuns is not particularly favorable, and we also obtain interesting algorithmic results on sorting.

Corollary 3.7. *There is an encoding scheme using at most* $\tau\lceil\lg\rho\rceil(1+o(1)) + 2\tau\lg\frac{n}{\tau} + \mathcal{O}(\tau) + o(n)$ *bits to encode a permutation* π *over* $[n]$ *covered by* τ *strict runs and by* $\rho \leq \tau$ *runs. It supports* $\pi(i)$ *and* $\pi^{-1}(i)$ *in time* $\mathcal{O}(1+\log\rho)$ *for any value of* $i \in [n]$.

Corollary 3.8. *We can sort a permutation of* $[n]$*, covered by* τ *strict runs and by* ρ *runs, and* HRuns *being the run lengths of the strict run heads, in time* $\mathcal{O}(n + \tau H(\texttt{HRuns})) = \mathcal{O}(n+\tau\log\rho)$*, which is worst-case optimal, in the comparison model, among all permutations sharing these* ρ*,* τ*, and* HRuns *values, such that* $\rho\log\tau = o(\tau H(\texttt{HRuns}))$.

3.3. Shuffled Sequences

Levcopoulos and Petersson [28] introduced the more sophisticated concept of partitions formed by interleaved runs, such as *Shuffled UpSequences* (SUS). We discuss here the advantage of considering permutations formed by shuffling a small number of runs.

Definition 3.9. A decomposition of a permutation π over $[n]$ into *Shuffled UpSequences* is a set of, not necessarily consecutive, subsequences of increasing numbers that have to be removed from π in order to reduce it to the empty sequence. The minimum number of shuffled upsequences in such a decomposition of π is noted σ, and the sequence of the lengths of the involved shuffled upsequences, in arbitrary order, is noted SUS.

For example, permutation $(\mathit{1}, \mathbf{6}, \mathit{2}, \mathbf{7}, \mathit{3}, \mathbf{8}, \mathit{4}, \mathbf{9}, \mathit{5}, \mathbf{10})$ contains $\sigma = 2$ shuffled upsequences of lengths $\texttt{SUS} = \langle 5, 5\rangle$, but $\rho = 5$ runs, all of length 2. Whereas the decomposition of a permutation into runs or strict runs can be computed in linear time, the decomposition into shuffled upsequences requires a bit more time. Fredman [16] gave an algorithm to compute the size of an optimal partition, claiming a worst case complexity of $\mathcal{O}(n\log n)$. In fact his algorithm is adaptive and takes $\mathcal{O}(n(1+\log\sigma))$ time. We give here a variant of his algorithm which computes the partition itself within the same complexity, and we achieve even better time on favorable sequences SUS.

Lemma 3.10. *Given a permutation* π *over* $[n]$ *covered by* σ *shuffled upsequences of lengths* SUS*, there is an algorithm finding such a partition in time* $\mathcal{O}(n(1+H(\texttt{SUS})))$.

Proof. Initialize a sequence $S_1 = (\pi(1))$, and a splay tree T [45] with the node (S_1), ordered by the rightmost value of the sequence contained by each node. For each further element $\pi(i)$, search for the sequence with the maximum ending point smaller than $\pi(i)$. If any, add $\pi(i)$ to this sequence, otherwise create a new sequence and add it to T. Fredman [16] already proved that this algorithm computes an optimal partition. The adaptive complexity results from the mere observation that the splay tree (a simple sorted array in Fredman's proof) contains at most σ elements, and that the node corresponding to a subsequence is accessed once per element in it. Hence the total access time is $\mathcal{O}(n(1+H(\texttt{SUS})))$ [45, Thm. 2]. ∎

The complete description of the permutation requires to encode the computation of both the partitioning algorithm and the sorting one, and this time the encoding cost of partitioning is as important as that of merging.

Theorem 3.11. *There is an encoding scheme using at most* $2n(1+H(\texttt{SUS}))+o(n\log\sigma)+\mathcal{O}(\sigma\log n)$ *bits to encode a permutation* π *over* $[n]$ *covered by* σ *shuffled upsequences of lengths* SUS*. It supports the operations* $\pi(i)$ *and* $\pi^{-1}(i)$ *in time* $\mathcal{O}(1+\log\sigma)$ *for any value of* $i\in[n]$*. If* i *is chosen uniformly at random in* $[n]$ *the average time is* $\mathcal{O}(1+H(\texttt{SUS})+\frac{\log\sigma}{\log\log n})$.

Proof. Partition the permutation π into σ shuffled upsequences using Lemma 3.10, resulting in a string S of length n over alphabet $[\sigma]$ which indicates for each element of the permutation π the label of the upsequence it belongs to. Encode S with a wavelet tree using Raman *et al.*'s compression for the bitmaps, so as to achieve $nH(\texttt{SUS})+o(n\log\sigma)$ bits of space and support retrieval of any $S[i]$, as well as symbol *rank* and *select* on S, in time $\mathcal{O}(1+\log\sigma)$ (§2). Store also an array $A[1,\sigma]$ so that $A[\ell]$ is the accumulated length of all the upsequences with label less than ℓ. Array A requires $\mathcal{O}(\sigma\log n)$ bits. Finally, consider the permutation π' formed by the upsequences taken in label order: π' has at most σ runs and hence can be encoded using $n(2+H(\texttt{SUS}))(1+o(1))+\mathcal{O}(\sigma\log n)$ bits using Thm. 3.2, as SUS in π corresponds to Runs in π'. This supports $\pi'(i)$ and $\pi'^{-1}(i)$ in time $\mathcal{O}(1+\log\sigma)$.

Now $\pi(i)=\pi'(A[S[i]]+rank_{S[i]}(S,i))$ can be computed in time $\mathcal{O}(1+\log\sigma)$. Similarly, $\pi^{-1}(i)=select_\ell(S,(\pi')^{-1}(i)-A[\ell])$, where ℓ is such that $A[\ell]<(\pi')^{-1}(i)\le A[\ell+1]$, can also be computed in $\mathcal{O}(1+\log\sigma)$ time. Thus the whole structure uses $2n(1+H(\texttt{SUS}))+o(n\log\sigma)+\mathcal{O}(\sigma\log n)$ bits and supports $\pi(i)$ and $\pi^{-1}(i)$ in time $\mathcal{O}(1+\log\sigma)$.

The obstacles to achieve the claimed average time are the operations on the wavelet tree of S, and the binary search in A. The former can be reduced to $\mathcal{O}(1+\frac{\log\sigma}{\log\log n})$ by using the improved wavelet tree representation by Ferragina *et al.* (§2). The latter is reduced to constant time by representing A with a bitmap $A'[1,n]$ with the bits set at the values $A[\ell]+1$, so that $A[\ell]=select_1(A',\ell)-1$, and the binary search is replaced by $\ell=rank_1(A',(\pi')^{-1}(i))$. With Raman *et al.*'s structure (§2), A' needs $\mathcal{O}(\sigma\log\frac{n}{\sigma})$ bits and operates in constant time. ∎

Again, we might prefer a simplified result when SUS has no interesting distribution, and we also achieve an improved result on sorting, better than the known $\mathcal{O}(n(1+\log\sigma))$.

Corollary 3.12. *There is an encoding scheme using at most* $2n\lg\sigma(1+o(1))+\sigma\lg\frac{n}{\sigma}+\mathcal{O}(\sigma)$ *bits to encode a permutation* π *over* $[n]$ *covered by* σ *shuffled upsequences. It supports the operations* $\pi(i)$ *and* $\pi^{-1}(i)$ *in time* $\mathcal{O}(1+\log\sigma)$ *for any value of* $i\in[n]$.

Corollary 3.13. *We can sort an array of length* n*, covered by* σ *shuffled upsequences of lenghts* SUS*, in time* $\mathcal{O}(n(1+H(\texttt{SUS})))$*, which is worst-case optimal, in the comparison model, among all permutations decomposable into* σ *shuffled upsequences of lenghts* SUS *such that* $\sigma\log n=o(nH(\texttt{SUS}))$.

4. Applications

4.1. Inverted Indexes

Consider a full-text inverted index which gives the word positions of any word in a text. This is a popular data structure for natural language text retrieval [3, 46], as it permits for example solving phrase queries without accessing the text. For each different text word, an increasing list of its text positions is stored.

Let n be the total number of words in a text collection $T[1,n]$ and ρ the vocabulary size (i.e., number of different words). An uncompressed inverted index requires $(\rho+n)\lceil \lg n \rceil$ bits. It has been shown [31] that, by δ-encoding the differences between consecutive entries in the inverted lists, the total space reduces to $nH_0(T) + \rho\lceil \lg n \rceil$, where $H_0(T)$ is the zero-order entropy of the text if seen as a sequence of words (§2). We note that the empirical law by Heaps [22], well accepted in Information Retrieval, establishes that ρ is small: $\rho = \mathcal{O}(n^\beta)$ for some constant $0 < \beta < 1$ depending on the text type.

Several successful methods to compress natural language text take words as symbols and use zero-order encoding, and thus the size they can achieve is lower bounded by $nH_0(T)$ [35]. If we add the differentially encoded inverted index in order to be able of searching the compressed text, the total space is at least $2nH_0(T)$.

Now, the concatenation of the ρ inverted lists can be seen as a permutation of $[n]$ with ρ runs, and therefore Thm. 3.2 lets us encode it in $n(2+H_0(T))(1+o(1))+\mathcal{O}(\rho \log n)$ bits. Within the same space we can add ρ numbers telling where the runs begin, in an array $V[1,\rho]$. Now, in order to retrieve the list of the i-th word, we simply obtain $\pi(V[i]), \pi(V[i]+1), \ldots, \pi(V[i+1]-1)$, each in $\mathcal{O}(1+\log \rho)$ time. Moreover we can extract any random position from a list, which enables binary-search-based strategies for list intersection [2, 42, 13]. In addition, we can also obtain a text passage from the (inverse) permutation: To find out $T[j]$, $\pi^{-1}(j)$ gives its position in the inverted lists, and a binary search on V finds the interval $V[i] \leq \pi^{-1}(j) < V[i+1]$, to output that $T[j] = i$th word, in $\mathcal{O}(1+\log \rho)$ time.

This result is very interesting, as it constitutes a true word-based *self-index* [39] (i.e., a compressed text index that contains the text). Similar results have been recently obtained with rather different methods [9, 11]. The cleanest one is to build a wavelet tree over T with compression [15], which achieves $nH_0(T) + o(n \log \rho) + \mathcal{O}(\rho \log n)$ bits of space, and permits obtaining $T[i]$, as well as extracting the jth element of the inverted list of the ith word with $select_i(T,j)$, all in time $\mathcal{O}(1+\frac{\log \rho}{\log \log n})$.

Yet, one advantage of our approach is that the extraction of ℓ consecutive entries $\pi^{-1}([i,i'])$ takes $\mathcal{O}(\ell(1+\log \frac{\rho}{\ell}))$ time if we do the process for all the entries as a block: Start at range $[i,i']$ at the root bitmap B, with position $p \leftarrow 0$, and bitmap size $s \leftarrow n$. Go down to both left and right children: to the left with $[i,i'] \leftarrow [rank_0(B,i), rank_0(B,i')]$, same p, and $s \leftarrow rank_0(B,s)$; to the right with $[i,i'] \leftarrow [rank_1(B,i), rank_1(B,i')]$, $p \leftarrow p + rank_0(B,s)$, and $s \leftarrow rank_1(B,s)$. Stop when the range $[i,i']$ becomes empty or when we reach a leaf, in which case report all answers $p+k$, $i \leq k \leq i'$. By representing the inverted list as π^{-1}, we can extract long inverted lists faster than the existing methods.

Corollary 4.1. *There exists a representation for a text $T[1,n]$ of integers in $[1,\rho]$ (regarded as word identifiers), with zero-order entropy H_0, that takes $n(2+H_0)(1+o(1))+\mathcal{O}(\rho \log n)$ bits of space, and can retrieve the text position of the jth occurrence of the ith text word, as well as the value $T[j]$, in $\mathcal{O}(1+\log \rho)$ time. It can also retrieve any range of ℓ successive occurrences of the ith text word in time $\mathcal{O}(\ell(1+\log \frac{\rho}{\ell}))$.*

We could, instead, represent the inverted list as π, so as to extract long text passages efficiently, but the wavelet tree representation can achieve the same result. Another interesting functionality that both representations share, and which is useful for other list intersection algorithms [6, 4], is that to obtain the first entry of a list which is larger than x. This is done with *rank* and *select* on the wavelet tree representation. In our permutation representation, we can also achieve it in $\mathcal{O}(1+\log \rho)$ time by finding out the position of a number x within a given run. The algorithm is similar to those in Thm. 3.2 that descend

to a leaf while maintaining the offset within the node, except that the decision on whether to descend left or right depends on the leaf we want to arrive at and not on the bitmap content (this is actually the algorithm to compute *rank* on binary wavelet trees [39]).

Finally, we note that our inverted index data structure supports in small time all the operations required to solve conjunctive queries on binary relations.

4.2. Suffix Arrays

Suffix arrays are used to index texts that cannot be handled with inverted lists. Given a text $T[1,n]$ of n symbols over an alphabet of size ρ, the *suffix* array $A[1,n]$ is a permutation of $[n]$ so that $T[A[i],n]$ is lexicographically smaller than $T[A[i+1],n]$. As suffix arrays take much space, several compressed data structures have been developed for them [39]. One of interest for us is the *Compressed Suffix Array (CSA)* of Sadakane [41]. It builds over a permutation Ψ of $[n]$, which satisfies $A[\Psi[i]] = (A[i] \bmod n) + 1$ (and thus lets us move virtually one position forward in the text) [20]. It turns out that, using just Ψ and $\mathcal{O}(\rho \log n)$ extra bits, one can (i) *count* the number of times a pattern $P[1,m]$ occurs in T using $\mathcal{O}(m \log n)$ applications of Ψ; (ii) *locate* any such occurrence using $\mathcal{O}(s)$ applications of Ψ, by spending $\mathcal{O}(\frac{n \log n}{s})$ extra bits of space; and (iii) *extract* a text substring $T[l,r]$ using at most $s + r - l$ applications of Ψ. Hence this is another self-index, and its main burden of space is that to represent permutation Ψ.

Sadakane shows that Ψ has at most ρ runs, and gives a representation that accesses $\Psi[i]$ in constant time by using $nH_0(T) + \mathcal{O}(n \log\log \rho)$ bits of space. It was shown later [39] that the space is actually $nH_k(T) + \mathcal{O}(n \log\log \rho)$ bits, for any $k \leq \alpha \log_\rho n$ and constant $0 < \alpha < 1$. Here $H_k(T) \leq H_0(T)$ is the kth order empirical entropy of T [33].

With Thm. 3.2 we can encode Ψ using $n(2 + H_0(T))(1 + o(1)) + \mathcal{O}(\rho \log n)$ bits of space, whose extra terms aside from entropy are better than Sadakane's. Those extra terms can be very significant in practice. The price is that the time to access Ψ is $\mathcal{O}(1 + \log \rho)$ instead of constant. On the other hand, an interesting extra functionality is that to compute Ψ^{-1}, which lets us move (virtually) one position backward in T. This allows, for example, displaying the text context around an occurrence without having to spend any extra space. Still, although interesting, the result is not competitive with recent developments [15, 30].

An interesting point is that Ψ contains $\tau \leq \min(n, nH_k(T) + \rho^k)$ strict runs, for any k [29]. Therefore, Cor. 3.7 lets us represent it using $\tau \lceil \lg \rho \rceil (1+o(1)) + 2\tau \lg \frac{n}{\tau} + \mathcal{O}(\tau) + o(n)$ bits of space. For k limited as above, this is at most $nH_k(T)(\lg \rho + 2 \lg \frac{1}{H_k(T)} + \mathcal{O}(1)) + o(n \log \rho)$ bits, which is similar to the space achieved by another self-index [29, 43], yet again it is slightly superseded by its time performance.

4.3. Iterated Permutation

Munro *et al.* [37] described how to represent a permutation π as the concatenation of its cycles, completed by a bitvector of n bits coding the lengths of the cycles. As the cycle representation is itself a permutation of $[n]$, we can use any of the permutation encodings described in §3 to encode it, adding the binary vector encoding the lengths of the cycles. It is important to note that, for a specific permutation π, the difficulty to compress its cycle encoding π' is not the same as the difficulty to encode the original permutation π.

Given a permutation π with c cycles of lengths $\langle n_1, \ldots, n_c \rangle$, there are several ways to encode it as a permutation π', depending on the starting point of each cycle ($\Pi_{i \in [c]} n_i$

choices) and the order of the cycles in the encoding ($c!$ choices). As a consequence, each permutation π with c cycles of lengths $\langle n_1, \ldots, n_c \rangle$ can be encoded by any of the $\Pi_{i \in [c]} i \times n_i$ corresponding permutations.

Corollary 4.2. *Any of the encodings from Theorems 3.2, 3.6 and 3.11 can be combined with an additional cost of at most $n + o(n)$ bits to encode a permutation π over $[n]$ composed of c cycles of lengths $\langle n_1, \ldots, n_c \rangle$ to support the operation $\pi^k(i)$ for any value of $k \in \mathbb{Z}$, in time and space function of the order in the permutation encoding of the cycles of π.*

The space "wasted" by such a permutation representation of the cycles of π is $\sum \lg n_i + c \lg c$ bits. To recover some of this space, one can define a canonical cycle encoding by starting the encoding of each cycle with its smallest value, and by ordering the cycles in order of their starting point. This canonical encoding always starts with a 1 and creates at least one shuffled upsequence of length c: it can be compressed as a permutation over $[n-1]$ with at least one shuffled upsequence of length $c+1$ through Thm 3.11.

4.4. Integer Functions

Munro and Rao [38] extended the results on permutations to arbitrary functions from $[n]$ to $[n]$, and to their iterated application $f^k(i)$, the function iterated k times starting at i. Their encoding is based on the decomposition of the function into a bijective part, represented as a permutation, and an injective part, represented as a forest of trees whose roots are elements of the permutation: the summary of the concept is that an integer function is just a "hairy permutation". Combining the representation of permutations from [37] with any representation of trees supporting the level-ancestor operator and an iterator of the descendants at a given level yields a representation of an integer function f using $(1+\varepsilon) n \lg n + \mathcal{O}(1)$ bits to support $f^k(i)$ in $\mathcal{O}(1 + |f^k(i)|)$ time, for any fixed ε, integer $k \in \mathbb{Z}$ and $i \in [n]$.

Janssen *et al.* [25] defined the *degree entropy* of an ordered tree T with n nodes, having n_i nodes with i children, as $H^*(T) = H(\langle n_1, n_2, \ldots \rangle)$, and proposed a succinct data structure for T using $nH^*(T) + \mathcal{O}(n(\lg\lg n)^2 / \lg n)$ bits to encode the tree and support, among others, the level-ancestor operator. Obviously, the definition and encoding can be generalized to a forest of k trees by simply adding one node whose k children are the roots of the k trees.

Encoding the injective parts of the function using Janssen *et al.*'s [25] succinct encoding, and the bijective parts of the function using one of our permutation encodings, yields a compressed representation of any integer function which supports its application and the application of its iterated variants in small time.

Corollary 4.3. *There is a representation of a function $f : [n] \to [n]$ that uses $n(1 + \lceil \lg \rho \rceil + H^*(T)) + o(n \lg n)$ bits to support $f^k(i)$ in $\mathcal{O}(\log \rho + |f^k(i)|)$ time, for any integer k and for any $i \in [n]$, where T is the forest representing the injective part of the function, and ρ is the number of runs in the bijective part of the function.*

5. Conclusion

Bentley and Yao [8], when introducing a family of search algorithms adaptive to the position of the element searched (aka the "unbounded search" problem), did so through the definition of a family of adaptive codes for unbounded integers, hence proving that the

link between algorithms and encodings was not limited to the complexity lower bounds suggested by information theory.

In this paper, we have considered the relation between the difficulty measures of adaptive sorting algorithms and some measures of "entropy" for compression techniques on permutations. In particular, we have shown that some concepts originally defined for adaptive sorting algorithms, such as runs and shuffled upsequences, are useful in terms of the compression of permutations; and conversely, that concepts originally defined for data compression, such as the entropy of the sets of sizes of runs, are a useful addition to the set of difficulty measures that one can consider in the study of adaptive algorithms.

It is easy to generalize our results on runs and strict runs to take advantage of permutations which are a mix of up and down runs or strict runs (e.g. $(1, 3, 5, 7, 9, \mathbf{10}, \mathbf{8}, \mathbf{6}, \mathbf{4}, \mathbf{2})$, with only a linear extra computational and/or space cost. The generalization of our results on shuffled upsequences to SMS [28], permutations containing mixes of subsequences sorted in increasing and decreasing orders (e.g. $(1, \mathbf{10}, 2, \mathbf{9}, 3, \mathbf{8}, 4, \mathbf{7}, 5, \mathbf{6})$) is sligthly more problematic, because it is NP hard to optimally decompose a permutation into such subsequences [26], but any approximation scheme [28] would yield a good encoding.

Refer to the associated technical report [7] for a longer version of this paper, in particular including all the proofs.

References

[1] D. Arroyuelo, G. Navarro, and K. Sadakane. Reducing the space requirement of LZ-index. In *Proc. 17th CPM*, LNCS 4009, pages 319–330, 2006.

[2] R. Baeza-Yates. A fast set intersection algorithm for sorted sequences. In *Proc. 15th CPM*, LNCS 3109, pages 400–408, 2004.

[3] R. Baeza-Yates and B. Ribeiro. *Modern Information Retrieval*. Addison-Wesley, 1999.

[4] J. Barbay, A. Golynski, J. I. Munro, and S. S. Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. *Theor. Comp. Sci.*, 2007.

[5] J. Barbay, M. He, J. I. Munro, and S. S. Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *Proc. 18th SODA*, pages 680–689, 2007.

[6] J. Barbay, A. López-Ortiz, and T. Lu. Faster adaptive set intersections for text searching. In *Proc. 5th WEA*, LNCS 4007, pages 146–157, 2006.

[7] J. Barbay and G. Navarro. Compressed representations of permutations, and applications. Technical Report TR/DCC-2008-18, Department of Computer Science (DCC), University of Chile, December 2008. `http://www.dcc.uchile.cl/TR/2008/TR_DCC-2008-018.pdf`.

[8] J. L. Bentley and A. C.-C. Yao. An almost optimal algorithm for unbounded searching. *Inf. Proc. Lett.*, 5(3):82–87, 1976.

[9] N. Brisaboa, A. Fariña, S. Ladra, and G. Navarro. Reorganizing compressed text. In *Proc. 31st SIGIR*, pages 139–146, 2008.

[10] D. Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, Canada, 1996.

[11] F. Claude and G. Navarro. Practical rank/select queries over arbitrary sequences. In *Proc. 15th SPIRE*, LNCS 5280, pages 176–187, 2008.

[12] C. Cool and D. Kim. Best sorting algorithm for nearly sorted lists. *Comm. ACM*, 23:620–624, 1980.

[13] J. Culpepper and A. Moffat. Compact set representation for information retrieval. In *Proc. 14th SPIRE*, pages 137–148, 2007.

[14] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Comp. Surv.*, 24(4):441–476, 1992.

[15] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. Compressed representations of sequences and full-text indexes. *ACM Trans. on Algorithms (TALG)*, 3(2):article 20, 2007.

[16] M. L. Fredman. On computing the length of longest increasing subsequences. *Discrete Math.*, 11:29–35, 1975.

[17] A. Golynski. Optimal lower bounds for rank and select indexes. In *Proc. 33th ICALP*, LNCS 4051, pages 370–381, 2006.
[18] A. Golynski, J. I. Munro, and S. S. Rao. Rank/select operations on large alphabets: a tool for text indexing. In *Proc. 17th SODA*, pages 368–373, 2006.
[19] R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th SODA*, pages 841–850, 2003.
[20] R. Grossi and J. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. on Computing*, 35(2):378–407, 2006.
[21] L. Guibas, E. McCreight, M. Plass, and J. Roberts. A new representation of linear lists. In *Proc. 9th STOC*, pages 49–60, 1977.
[22] H. Heaps. *Information Retrieval - Computational and Theoretical Aspects*. Academic Press, NY, 1978.
[23] T. Hu and A. Tucker. Optimal computer-search trees and variable-length alphabetic codes. *SIAM J. of Applied Mathematics*, 21:514–532, 1971.
[24] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.*, 40(9):1090–1101, 1952.
[25] J. Jansson, K. Sadakane, and W.-K. Sung. Ultra-succinct representation of ordered trees. In *Proc. 18th SODA*, pages 575–584, 2007.
[26] A. E. Kézdy, H. S. Snevily, and C. Wang. Partitioning permutations into increasing and decreasing subsequences. *J. Comb. Theory Ser. A*, 73(2):353–359, 1996.
[27] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 2nd edition, 1998.
[28] C. Levcopoulos and O. Petersson. Sorting shuffled monotone sequences. *Inf. Comp.*, 112(1):37–50, 1994.
[29] V. Mäkinen and G. Navarro. Succinct suffix arrays based on run-length encoding. *Nordic J. of Computing*, 12(1):40–66, 2005.
[30] V. Mäkinen and G. Navarro. Implicit compression boosting with applications to self-indexing. In *Proc. 14th SPIRE*, LNCS 4726, pages 214–226, 2007.
[31] V. Mäkinen and G. Navarro. Rank and select revisited and extended. *Theor. Comp. Sci.*, 387(3):332–347, 2007.
[32] H. Mannila. Measures of presortedness and optimal sorting algorithms. In *IEEE Trans. Comput.*, volume 34, pages 318–325, 1985.
[33] G. Manzini. An analysis of the Burrows-Wheeler transform. *J. of the ACM*, 48(3):407–430, 2001.
[34] K. Mehlhorn. Sorting presorted files. In *Proc. 4th GI-Conference on Theoretical Computer Science*, LNCS 67, pages 199–212, 1979.
[35] E. Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast and flexible word searching on compressed text. *ACM Trans. on Information Systems (TOIS)*, 18(2):113–139, 2000.
[36] I. Munro. Tables. In *Proc. 16th FSTTCS*, LNCS 1180, pages 37–42, 1996.
[37] J. I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations. In *Proc. 30th ICALP*, LNCS 2719, pages 345–356, 2003.
[38] J. I. Munro and S. S. Rao. Succinct representations of functions. In *Proc. 31st ICALP*, LNCS 3142, pages 1006–1015, 2004.
[39] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Comp. Surv.*, 39(1):article 2, 2007.
[40] R. Raman, V. Raman, and S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proc. 13th SODA*, pages 233–242, 2002.
[41] K. Sadakane. New text indexing functionalities of the compressed suffix arrays. *J. of Algorithms*, 48(2):294–313, 2003.
[42] P. Sanders and F. Transier. Intersection in integer inverted indices. In *Proc. 9th ALENEX*, 2007.
[43] J. Sirén, N. Välimäki, V. Mäkinen, and G. Navarro. Run-length compressed indexes are superior for highly repetitive sequence collections. In *Proc. 15th SPIRE*, LNCS 5280, pages 164–175, 2008.
[44] S. S. Skiena. Encroaching lists as a measure of presortedness. *BIT*, 28(4):775–784, 1988.
[45] D. Sleator and R. Tarjan. Self-adjusting binary search trees. *J. of the ACM*, 32(3):652–686, 1985.
[46] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes*. Morgan Kaufmann, 2nd edition, 1999.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 123–134
www.stacs-conf.org

ON THE AVERAGE COMPLEXITY OF MOORE'S STATE MINIMIZATION ALGORITHM

FRÉDÉRIQUE BASSINO [1] AND JULIEN DAVID [2] AND CYRIL NICAUD [2]

[1] LIPN UMR 7030, Université Paris 13 - CNRS, 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France.
E-mail address: Frederique.Bassino@lipn.univ-paris13.fr

[2] Institut Gaspard Monge, Université Paris Est, 77454 Marne-la-Vallée Cedex 2, France
E-mail address: Julien.David@univ-paris-est.fr,Cyril.Nicaud@univ-paris-est.fr

Abstract. We prove that, for any arbitrary finite alphabet and for the uniform distribution over deterministic and accessible automata with n states, the average complexity of Moore's state minimization algorithm is in $\mathcal{O}(n \log n)$. Moreover this bound is tight in the case of unary automata.

1. Introduction

Deterministic automata are a convenient way to represent regular languages that can be used to efficiently perform most of usual computations involving regular languages. Therefore finite state automata appear in many fields of computer science, such as linguistics, data compression, bioinformatics, etc. To a given regular language one can associate a unique smallest deterministic automaton, called its minimal automaton. This canonical representation of regular languages is compact and provides an easy way to check equality. As a consequence, state minimization algorithms that compute the minimal automaton of a regular language, given by a deterministic automaton, are fundamental.

Moore proposed a solution [15] that can be seen as a sequence of partition refinements. Starting from a partition of the set of states, of size n, into two parts, successive refinements lead to a partition whose elements are the subsets of indistinguishable sets, that can be merged to form a smaller automaton recognizing the same language. As there are at most $n-2$ such refinements, each of them requiring a linear running time, the worst-case complexity of Moore's state minimization algorithm is quadratic. Hopcroft's state minimization algorithm [11] also uses partition refinements to compute the minimal automaton, selecting carefully the parts that are split at each step. Using suitable data structures, its worst-case complexity is in $\mathcal{O}(n \log n)$. It is the best known minimization algorithm, and therefore it has been intensively studied, see [1, 5, 9, 12] for instance. Finally Brzozowski's algorithm [6, 7] is different from the other ones. Its inputs may be non-deterministic automata.

1998 ACM Subject Classification: F.2 Analysis of algorithms and problem complexity.
Key words and phrases: finite automata, state minimization, Moore's algorithm, average complexity.

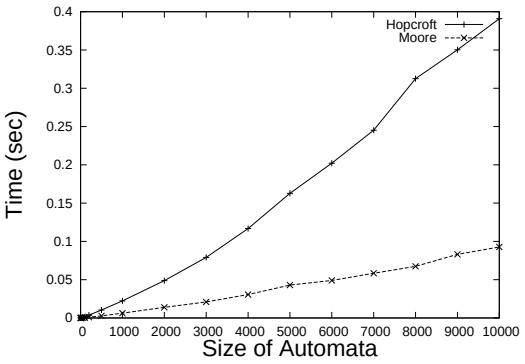


Figure 1: Time complexity of Moore's and Hopcroft's algorithms

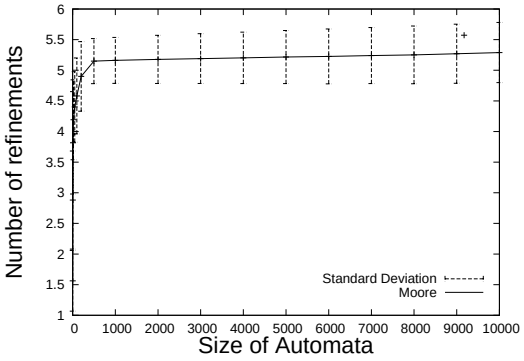


Figure 2: Number of partition refinements in Moore's algorithm

The results of Fig.1 and Fig.2 were obtained with the C++ library REGAL (available at: `http://regal.univ-mlv.fr/`) to randomly generate deterministic accessible automata [2, 3, 4]. Each value is computed from 20 000 automata over a 2-letter alphabet.

It is based on two successive determinization steps, and though its worst-case complexity is proved to be exponential, it has been noticed that it is is often sub-exponential in practice. The reader is invited to consult [17], which presents a taxonomy of minimization algorithms, for a more exhaustive list.

In this paper we study the average time complexity of Moore's algorithm. From an experimental point of view, the average complexity of Moore's algorithm seems to be smaller than the complexity of Hopcroft's algorithm (Fig.1) and the number of partition refinements increases very slowly as the size of the input grows (Fig.2). In the following we mainly prove that in average, for the uniform distribution, Moore's algorithm performs only $\mathcal{O}(\log n)$ refinements, thus its average complexity is in $\mathcal{O}(n \log n)$.

After briefly recalling the basics of minimization of automata in Section 2, we prove in Section 3 that the average time complexity of Moore's algorithm is $\mathcal{O}(n \log n)$ and show in Section 4 that this bound is tight when the alphabet is unary. The paper closes with a short discussion about generalizations of our main theorem to Bernoulli distributions and to incomplete automata in Section 5, and the presentation of a conjecture based on the slow growth of the number of refinements (Fig.2 when the alphabet is not unary in Section 6.

2. Preliminaries

This section is devoted to basic notions related to the minimization of automata. We refer the reader to the literature for more details about minimization of automata [10, 14, 18]. We only record a few definitions and results that will be useful for our purpose.

2.1. Finite automata

A *finite deterministic automaton* $\mathcal{A}$ is a quintuple $\mathcal{A} = (A, Q, \cdot, q_0, F)$ where Q is a finite set of *states*, A is a finite set of *letters* called *alphabet*, the *transition function* $\cdot$ is a mapping from $Q \times A$ to Q, $q_0 \in Q$ is the *initial state* and $F \subset Q$ is the set of final states. An automaton is *complete* when its transition function is total. The transition function can be extended by morphism to all words of A^*: $p \cdot \varepsilon = p$ for any $p \in Q$ and for any $u, v \in A^*$,

$p \cdot (uv) = (p \cdot u) \cdot v$. A word $u \in A^*$ is *recognized* by an automaton when $p \cdot u \in F$. The set of all words recognized by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. An automaton is *accessible* when for any state $p \in Q$, there exists a word $u \in A^*$ such that $q_0 \cdot u = p$.

A *transition structure* is an automaton where the set of final states is not specified. Given such a transition structure $\mathcal{T} = (A, Q, \cdot, q_0)$ and a subset F of Q, we denote by $(\mathcal{T}, F)$ the automaton $(A, Q, \cdot, q_0, F)$. For a given deterministic and accessible transition structure with n states there are exactly 2^n distinct deterministic and accessible automata that can be built from this transition structure. Each of them corresponds to a choice of set of final states.

In the following we only consider complete accessible deterministic automata and complete accessible deterministic transition structures, except in the presentation of the generalizations of the main theorem in Section 5. Consequently these objects will often just be called respectively *automata* or *transition structures*. The set Q of states of an n-state transition structure will be denoted by $\{1, \cdots, n\}$.

2.2. Myhill-Nerode equivalence

Let $\mathcal{A} = (A, Q, \cdot, q_0, F)$ be an automaton. For any nonnegative integer i, two states $p, q \in Q$ are *i-equivalent*, denoted by $p \sim_i q$, when for all words u of length less than or equal to i, $[\![p \cdot u \in F]\!] = [\![q \cdot u \in F]\!]$, where the Iverson bracket $[\![Cond]\!]$ is equal to 1 if the condition $Cond$ is satisfied and 0 otherwise. Two states are *equivalent* when for all $u \in A^*$, $[\![p \cdot u \in F]\!] = [\![q \cdot u \in F]\!]$. This equivalence relation is called *Myhill-Nerode equivalence*. An equivalence relation $\equiv$ defined on the set of states Q of a deterministic automaton is said to be *right invariant* when

$$\text{for all } u \in A^* \text{ and all } p, q \in Q, \quad p \equiv q \Rightarrow p \cdot u \equiv q \cdot u.$$

The following proposition [10, 14, 18] summarizes the properties of Myhill-Nerode equivalence that will be used in the next sections.

Proposition 2.1. *Let $\mathcal{A} = (A, Q, \cdot, q_0, F)$ be a deterministic automaton with n states. The following properties hold:*

1. *For all $i \in \mathbb{N}$, $\sim_{i+1}$ is a partition refinement of $\sim_i$, that is, for all $p, q \in Q$, if $p \sim_{i+1} q$ then $p \sim_i q$.*
2. *For all $i \in \mathbb{N}$ and for all $p, q \in Q$, $p \sim_{i+1} q$ if and only if $p \sim_i q$ and for all $a \in A$, $p \cdot a \sim_i q \cdot a$.*
3. *If for some $i \in \mathbb{N}$ $(i+1)$-equivalence is equal to i-equivalence then for every $j \geq i$, j-equivalence is equal to Myhill-Nerode equivalence.*
4. *$(n-2)$-equivalence is equal to Myhill-Nerode equivalence.*
5. *Myhill-Nerode equivalence is right invariant.*

Let $\mathcal{A} = (A, Q, \cdot, q_0, F)$ be an automaton and $\equiv$ be a right invariant equivalence relation on Q. The *quotient automaton* of $\mathcal{A}$ by $\equiv$ is the automaton

$$(\mathcal{A}/\!\equiv) = (A, Q/\!\equiv, *, [q_0], \{[f], f \in F\}),$$

where $Q/\!\equiv$ is the set of equivalent classes, $[q]$ is the class of $q \in Q$, and $*$ is defined for any $a \in A$ and any $q \in Q$ by $[q] * a = [q \cdot a]$. The correctness of this definition relies on the right invariance of the equivalence relation $\equiv$.

Theorem 2.2. *For any complete, accessible and deterministic automaton $\mathcal{A}$, the automaton $\mathcal{A}/\sim$ is the unique smallest automaton (in terms of the number of states) that recognizes the same language as the automaton $\mathcal{A}$. It is called the* minimal automaton *of $L(\mathcal{A})$.*

The uniqueness of the minimal automaton is up to labelling of the states. Theorem 2.2 shows that the minimal automaton is a fundamental notion in language theory: it is the most space efficient representation of a regular language by a deterministic automaton, and the uniqueness defines a bijection between regular language and minimal automata. For instance, to check whether two regular languages are equal, one can compare their minimal automata. It is one of the motivations for the algorithmic study of the computation of the minimal automaton of a language.

2.3. Moore's state minimization algorithm

In this section we describe the algorithm due to Moore [15] which computes the minimal automaton of a regular language represented by a deterministic automaton. Recall that Moore's algorithm builds the partition of the set of states corresponding to Myhill-Nerode equivalence. It mainly relies on properties (2) and (3) of Proposition 2.1: the partition π is initialized according to the 0-equivalence $\sim_0$, then at each iteration the partition corresponding to the $(i+1)$-equivalence $\sim_{i+1}$ is computed from the one corresponding to the i-equivalence $\sim_i$ using property (2). The algorithm halts when no new partition refinement is obtained, and the result is Myhill-Nerode equivalence according to property (3). The minimal automaton can then be computed from the resulting partition since it is the quotient automaton by Myhill-Nerode equivalence.

Algorithm 1: Moore

```
1  if F = ∅ then
2  |  return (A, {1}, ∗, 1, ∅)
3  end
4  if F = {1, ··· , n} then
5  |  return (A, {1}, ∗, 1, {1})
6  end
7  forall p ∈ {1, ··· , n} do
8  |  π'[p] = [[p ∈ F]]
9  end
10 π = undefined
11 while π ≠ π' do
12 |  π = π'
13 |  compute the partition π' s.t.
   |  π'[p] = π'[q] iff π[p] = π[q]
14 |  and ∀a ∈ A π[p · a] = π[q · a]
15 end
16 return the quotient of A by π
```

Algorithm 2: Computing π' from π

```
1  forall p ∈ {1, ··· , n} do
2  |  s[p] = (π[p], π[p · a_1], ··· , π[p · a_k])
3  end
4  compute the permutation σ that sorts the
   states according to s[]
5  i = 0
6  π'[σ(1)] = i
7  forall p ∈ {2, ··· , n} do
8  |  if s[p] ≠ s[p − 1] then  i = i + 1
9  |  π'[σ(p)] = i
10 end
11 return π'
```

In the description of Moore's algorithm, $*$ denotes the function such that $1 * a = 1$ for all $a \in A$. Lines 1-6 correspond to the special cases where $F = \emptyset$ or $F = Q$. In the process, π' is the new partition and π the former one. Lines 7-9 consists of the initialization of π' to the partition of $\sim_0$, π is initially undefined. Lines 11-14 are the main loop of the algorithm

where π is set to π' and the new π' is computed. Line 13 is described more precisely in the algorithm on the right: with each state p is associated a $k+1$-uple $s[p]$ such that two states should be in the same part in π' when they have the same $k+1$-uple. The matches are found by sorting the states according to their associated string.

The worst-case time complexity of Moore's algorithm is in $\mathcal{O}(n^2)$. The following result is a more precise statement about the worst-case complexity of this algorithm that will be used in the proof of the main theorem (Theorem 3.1). For sake of completeness we also give a justification of this statement.

For any integer $n \geq 1$ and any $m \in \{0, \cdots, n-2\}$, we denote by $\mathcal{A}_n^{(m)}$ the set of automata with n states for which m is the smallest integer such that the m-equivalence $\sim_m$ is equal to Myhill-Nerode equivalence. We also denote by $\textsc{Moore}(\mathcal{A})$ the number of iterations of the main loop when Moore's algorithm is applied to the automaton $\mathcal{A}$,

Lemma 2.3. *For any automaton $\mathcal{A}$ of $\mathcal{A}_n^{(m)}$,*

- *the number of iterations $\textsc{Moore}(\mathcal{A})$ of the main loop in Moore's algorithm is at most equal to $m+1$ and always less than or equal to $n-1$.*
- *the worst-case time complexity $\mathcal{W}(\mathcal{A})$ of Moore's algorithm is in $\Theta((m+1)n)$, where the Θ is uniform for $m \in \{0, \cdots, n-2\}$, or equivalently there exist two positive real numbers C_1 and C_2 independent of n and m such that $C_1(m+1)n \leq \mathcal{W}(\mathcal{A}) \leq C_2(m+1)n$.*

Proof. The result holds since the loop is iterated exactly $m+1$ times when the set F of final states is neither empty nor equal to $\{1, \cdots, n\}$. Moreover from property (4) of Proposition 2.1 the integer m is less than or equal to $n-2$. If F is empty or equal to $\{1, \cdots, n\}$, then necessarily $m = 0$, and the time complexity of the determination of the size of F is $\Theta(n)$.

The initialization and the construction of the quotient are both done in $\Theta(n)$. The complexity of each iteration of the main loop is in $\Theta(n)$: this can be achieved classically using a lexicographic sort algorithm. Moreover in this case the constants C_1 and C_2 do not depend on m, proving the uniformity of both the upper and lower bounds. ∎

Note that Lemma 2.3 gives a proof that the worst-case complexity of Moore's algorithm is in $\mathcal{O}(n^2)$, as there are no more than $n-1$ iterations in the process of the algorithm.

2.4. Probabilistic model

The choice of the distribution is crucial for average case analysis of algorithms. Here we are considering an algorithm that builds the minimal automaton of the language recognized by a given accessible deterministic and complete one. We focus our study on the average complexity of this algorithm for the uniform distribution over accessible deterministic and complete automata with n states, and as n tends toward infinity. Note that for the uniform distribution over automata with n states, the probability for a given set to be the set of final states is equal to $1/2^n$. Therefore the probability that all states are final (or non-final) is exponentially unlikely. Some extensions of the main result to other distributions are given in Section 5.

The general framework of the average case analysis of algorithms [8] is based on the enumeration properties of studied objects, most often given by generating functions. For accessible and deterministic automata, this first step is already not easy. Although the

asymptotic of the number of such automata is known, it can not be easily handled: a result from Korshunov [13], rewritten in terms of Stirling numbers of the second kind in [2] and generalized to possibly incomplete automata in [4], is that the number of accessible and deterministic automata with n states is asymptotically equal to $\alpha\beta^n n^{(|A|-1)n}$ where α and β are constants depending on the cardinality $|A|$ of the alphabet, and α depends on whether we are considering complete automata or possibly incomplete automata.

Here some good properties of Myhill-Nerode equivalence allow us to work independently and uniformly on each transition structure. In this way the enumeration problem mentioned above can be avoided. Nevertheless it should be necessary to enumerate some subsets of this set of automata in order to obtain a more precise result. One refers the readers to the discussion of Section 6 for more details.

3. Main result

This section is devoted to the statement and the proof of the main theorem.

Theorem 3.1. *For any fixed integer $k \geq 1$ and for the uniform distribution over the accessible deterministic and complete automata of size n over a k-letter alphabet, the average complexity of Moore's state minimization algorithm is $\mathcal{O}(n \log n)$.*

Note that this bound is independent of k, the size of the alphabet considered. Moreover, as we shall see in Section 4, it is tight in the case of a unary alphabet.

Before proving Theorem 3.1 we introduce some definitions and preliminary results. Let $\mathcal{T}$ be a fixed transition structure with n states and ℓ be an integer such that $1 \leq \ell < n$. Let p, q, p', q' be four states of $\mathcal{T}$ such that $p \neq q$ and $p' \neq q'$. We define $\mathcal{F}_\ell(p, q, p', q')$ as the set of sets of final states F for which in the automaton $(\mathcal{T}, F)$ the states p and q are $(\ell-1)$-equivalent, but not ℓ-equivalent, because of a word of length ℓ mapping p to p' and q to q' where p' and q' are not 0-equivalent. In other words $\mathcal{F}_\ell(p, q, p', q')$ is the following set:

$$\mathcal{F}_\ell(p, q, p', q') = \{F \subset \{1, \cdots, n\} \mid \text{for the automaton } (\mathcal{T}, F),\ p \sim_{\ell-1} q, [\![p' \in F]\!] \neq [\![q' \in F]\!], \\ \exists u \in A^\ell,\ p \cdot u = p' \text{ and } q \cdot u = q'\}$$

Note that when ℓ grows, the definition of $\mathcal{F}_\ell$ is more constrained and consequently fewer non-empty sets $\mathcal{F}_\ell$ exist.

From the previous set $\mathcal{F}_\ell(p, q, p', q')$ one can define the undirected graph $\mathcal{G}_\ell(p, q, p', q')$, called the *dependency graph*, as follows:

- its set of vertices is $\{1, \cdots, n\}$, the set of states of $\mathcal{T}$;
- there is an edge (s, t) between two vertices s and t if and only if for all $F \in \mathcal{F}_\ell(p, q, p', q')$, $[\![s \in F]\!] = [\![t \in F]\!]$.

The dependency graph contains some information that is a basic ingredient of the proof: it is a convenient representation of necessary conditions for a set of final states to be in $\mathcal{F}_\ell(p, q, p', q')$, that is, for Moore's algorithm to require more than ℓ iterations because of p, q, p' and q'. These necessary conditions will be used to give an upper bound on the cardinality of $\mathcal{F}_\ell(p, q, p', q')$ in Lemma 3.3.

Lemma 3.2. *For any integer $\ell \in \{1, \cdots, n-1\}$ and any states $p, q, p', q' \in \{1, \cdots, n\}$ with $p \neq q$, $p' \neq q'$ such that $\mathcal{F}_\ell(p, q, p', q')$ is not empty, there exists an acyclic subgraph of $\mathcal{G}_\ell(p, q, p', q')$ with ℓ edges.*

Proof. If $\mathcal{F}_\ell(p,q,p',q')$ is not empty, let $u = u_1 \cdots u_\ell$ with $u_i \in A$ be the smallest (for the lexicographic order) word of length ℓ such that $p \cdot u = p'$ and $q \cdot u = q'$. Note that every word u of length ℓ such that $p \cdot u = p'$ and $q \cdot u = q'$ can be used. But a non-ambiguous choice of this word u guarantees a complete description of the construction.

For every $i \in \{0, \cdots, \ell-1\}$, let $G_{\ell,i}$ be the subgraph of $\mathcal{G}_\ell(p,q,p',q')$ whose edges are defined as follows. An edge (s,t) is in $\mathcal{G}_{\ell,i}$ if and only if there exists a prefix v of u of length less than or equal to i such that $s = p \cdot v$ and $t = q \cdot v$. In other words the edges of $\mathcal{G}_{\ell,i}$ are exactly the edges $(p \cdot v, q \cdot v)$ between the states $p \cdot v$ and $q \cdot v$ where v ranges over the prefixes of u of length less than or equal to i. Such edges belong to $\mathcal{G}_\ell(p,q,p',q')$ since $p \sim_{\ell-1} q$. Moreover, the graphs $(G_{\ell,i})_{0 \le i \le \ell-1}$ have the following properties:

(1) For each $i \in \{0, \cdots, \ell-2\}$, $G_{\ell,i}$ is a strict subgraph of $G_{\ell,i+1}$. The graph $G_{\ell,i+1}$ is obtained from $G_{\ell,i}$ by adding an edge from $p \cdot w$ to $q \cdot w$, where w is the prefix of u of length $i+1$. This edge does not belong to $G_{\ell,i}$, for otherwise there would exist a strict prefix z of w such that either $p \cdot z = p \cdot w$ and $q \cdot z = q \cdot w$ or $p \cdot z = q \cdot w$ and $q \cdot z = p \cdot w$. In this case, let w' be the word such that $u = ww'$, then either $p \cdot zw' = p'$ and $q \cdot zw' = q'$ or $p \cdot zw' = q'$ and $q \cdot zw' = p'$. Therefore there would exist a word of length less than ℓ, zw', such that, for $F \in \mathcal{F}_\ell(p,q,p',q')$, $[\![p \cdot zw' \in F]\!] \neq [\![q' \cdot zw' \in F]\!]$ which is not possible since $p \sim_{\ell-1} q$ and $\mathcal{F}_\ell(p,q,p',q')$ is not empty. Hence this edge is a new one.

(2) For each $i \in \{0, \cdots, \ell-1\}$, $G_{\ell,i}$ contains $i+1$ edges. It is a consequence of property (1), since $G_{\ell,0}$ has only one edge between p and q.

(3) For each $i \in \{0, \cdots, \ell-1\}$, $G_{\ell,i}$ contains no loop. Indeed $p \cdot v \neq q \cdot v$ for any prefix of u since $p \not\sim q$ for any automaton $(\mathcal{T}, F)$ with $F \in \mathcal{F}_\ell(p,q,p',q')$, which is not empty.

(4) For each $i \in \{0, \cdots, \ell-1\}$, if there exists a path in $G_{\ell,i}$ from s to t, then $s \sim_{\ell-1-i} t$ in every automata $(\mathcal{T}, F)$ with $F \in \mathcal{F}_\ell(p,q,p',q')$. This property can be proved by induction.

We claim that every $G_{\ell,i}$ is acyclic. Assume that it is not true, and let $j \ge 1$ be the smallest integer such that $G_{\ell,j}$ contains a cycle. By property (1), $G_{\ell,j}$ is obtained from $G_{\ell,j-1}$ by adding an edge between $p \cdot w$ and $q \cdot w$ where w is the prefix of length j of u. As $G_{\ell,j-1}$ is acyclic, this edge forms a cycle in $G_{\ell,j}$. Hence in $G_{\ell,j-1}$ there already exists a path between $p \cdot w$ and $q \cdot w$. Therefore by property (4) $p \cdot w \sim_{\ell-j} q \cdot w$ in any automaton $(\mathcal{T}, F)$ with $F \in \mathcal{F}_\ell(p,q,p',q')$. Let w' be the word such that $u = ww'$. The length of w' is $\ell - j$, hence $p \cdot u$ and $q \cdot u$ are both in F or both not in F, which is not possible since $F \in \mathcal{F}_\ell(p,q,p',q')$.

Thus $G_{\ell,\ell-1}$ is an acyclic subgraph of $\mathcal{G}_\ell(p,q,p',q')$ with ℓ edges according to property (2), which concludes the proof. ∎

Lemma 3.3. *Given a transition structure $\mathcal{T}$ of size $n \ge 1$ and an integer ℓ with $1 \le \ell < n$, for all states p, q, p', q' of $\mathcal{T}$ with $p \neq q$ and $p' \neq q'$ the following result holds:*

$$|\mathcal{F}_\ell(p,q,p',q')| \le 2^{n-\ell}.$$

Proof. If $\mathcal{F}_\ell(p,q,p',q')$ is empty, the result holds. Otherwise, from Lemma 3.2, there exists an acyclic subgraph G of $\mathcal{G}_\ell(p,q,p',q')$ with ℓ edges. Let m be the number of connected components of G that are not reduced to a single vertex. The states in such a component are either all final or all non-final. Therefore there are at most m choices to make to determine whether the states in those components are final or non-final. As the graph G is acyclic, there are exactly $m + \ell$ vertices that are not isolated in G. Hence there are at

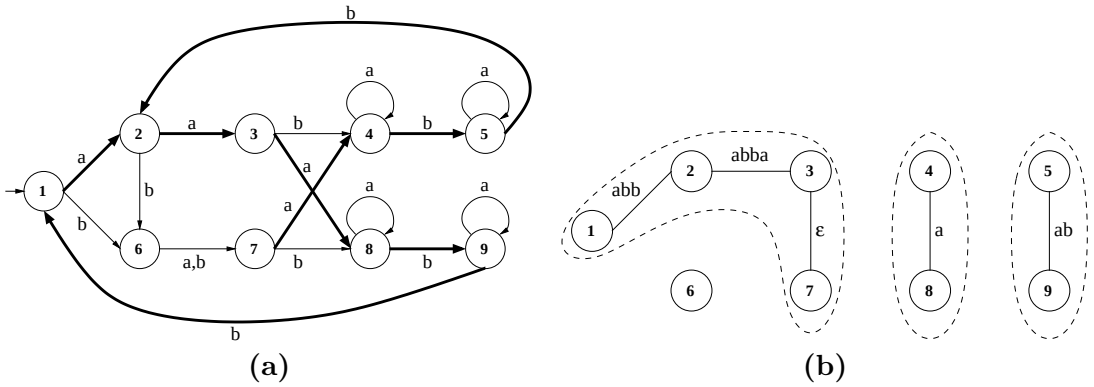


Figure 3: Illustration of the proof of Lemma 3.2 for $n = 9$, $\ell = 5$, $p = 3$, $q = 7$, $p' = 3$ and $q' = 8$ on a given transition structure. **(a)** $u = abbaa$ is the smallest word of length 5, for the lexicographic order, such that $3 \cdot u = 3$ and $7 \cdot u = 8$. The set $\mathcal{F}_5(3, 7, 3, 8)$ is not empty, as it contains $\{4, 8\}$. The bold transitions are the ones followed when reading u from p and from q. **(b)** The construction of an acyclic subgraph of $\mathcal{G}_5(3, 7, 3, 8)$ with 5 edges. To each strict prefix v of $u = abbaa$ is associated an edge between $3 \cdot v$ and $7 \cdot v$. It encodes some necessary conditions for a set of final states F to be in $\mathcal{F}_5(3, 7, 3, 8)$, as two states in the same connected component must be either both final or both not final.

most $2^m 2^{n-(m+\ell)} = 2^{n-\ell}$ elements in $\mathcal{F}_\ell(p, q, p', q')$: 2^m corresponds to the possible choices for the connected components and $2^{n-(m+\ell)}$ to the choices for the isolated vertices. ∎

Proposition 3.4. *Let $k \geq 1$. There exists a positive real constant C such that for any positive integer n and any deterministic and complete transition structure $\mathcal{T}$ of size n over a k-letter alphabet, for the uniform distribution over the sets F of final states, the average number of iterations of the main loop of Moore's algorithm applied to $(\mathcal{T}, F)$ is upper bounded by $C \log n$.*

Proof. Let $\mathcal{T}$ be a deterministic and complete transition structure of size n over a k-letter alphabet. Denote by $\mathcal{F}^{\geq \ell}$ the set of sets F of final states such that the execution of Moore's algorithm on $(\mathcal{T}, F)$ requires more than ℓ iterations or equivalently such that $(\mathcal{T}, F) \in \mathcal{A}_n^{(m)}$ with $m \geq \ell$ (see Section 2.3 for notation).

A necessary condition for F to be in $\mathcal{F}^{\geq \ell}$ is that there exist two states p and q with $p \neq q$ and such that $p \sim_{\ell-1} q$ and $p \not\sim_\ell q$. Therefore there exists a word u of length ℓ such that $[\![p \cdot u]\!] \neq [\![q \cdot u]\!]$. Hence $F \in \mathcal{F}_\ell(p, q, p \cdot u, q \cdot u)$ and

$$\mathcal{F}^{\geq \ell} = \bigcup_{\substack{p,q,p',q' \in \{1,\cdots,n\} \\ p \neq q,\ p' \neq q'}} \mathcal{F}_\ell(p, q, p', q').$$

In this union the sets $\mathcal{F}_\ell(p, q, p', q')$ are not disjoint, but this characterization of $\mathcal{F}^{\geq \ell}$ is precise enough to obtain a useful upper bound of the cardinality of $\mathcal{F}^{\geq \ell}$. From the description of $\mathcal{F}^{\geq \ell}$ we get

$$|\mathcal{F}^{\geq \ell}| \leq \sum_{\substack{p,q,p',q' \in \{1,\cdots,n\} \\ p \neq q,\ p' \neq q'}} |\mathcal{F}(p, q, p', q')|,$$

and using Lemma 3.3 and estimating the number of choices of the four points p, q, p', q', we have

$$|\mathcal{F}^{\geq \ell}| \leq n(n-1)\, n(n-1) 2^{n-\ell} \leq n^4 2^{n-\ell}. \tag{3.1}$$

For a fixed integer ℓ and for the uniform distribution over the sets F of final states, the average number of iterations of the main loop of Moore's algorithm is

$$\frac{1}{2^n} \sum_{F \subset \{1,\ldots,n\}} \textsc{Moore}(\mathcal{T}, F) = \frac{1}{2^n} \sum_{F \in \mathcal{F}^{<\ell}} \textsc{Moore}(\mathcal{T}, F) + \frac{1}{2^n} \sum_{F \in \mathcal{F}^{\geq \ell}} \textsc{Moore}(\mathcal{T}, F),$$

where $\mathcal{F}^{<\ell}$ is the complement of $\mathcal{F}^{\geq \ell}$ in the set of all subsets of states. Moreover by Lemma 2.3, for any $F \in \mathcal{F}^{<\ell}$, $\textsc{Moore}(\mathcal{T}, F) \leq \ell$. Therefore, since $|\mathcal{F}^{<\ell}| \leq 2^n$

$$\frac{1}{2^n} \sum_{F \in \mathcal{F}^{\leq \ell}} \textsc{Moore}(\mathcal{T}, F) \leq \ell$$

Using Lemma 2.3 again to give an upper bound for $\textsc{Moore}(\mathcal{T}, F)$ when $F \in \mathcal{F}^{\geq \ell}$ and the estimate of $|\mathcal{F}^{\geq \ell}|$ given by Equation 3.1 we have

$$\frac{1}{2^n} \sum_{F \subset \mathcal{F}^{\geq \ell}} \textsc{Moore}(\mathcal{T}, F) \leq n^5 2^{-\ell}.$$

Finally, choosing $\ell = \lceil 5 \log_2 n \rceil$, we obtain that there exists positive real C such that

$$\frac{1}{2^n} \sum_{F \subset \{1,\ldots,n\}} \textsc{Moore}(\mathcal{T}, F) \leq \lceil 5 \log_2 n \rceil + n^5 2^{-\lceil 5 \log_2 n \rceil} \leq C \log n,$$

concluding the proof. ∎

Now we prove Theorem 3.1:

Proof of the main theorem: Let $\mathcal{T}_n$ denote the set of deterministic, accessible and complete transition structures with n states. For a transition structure $\mathcal{T} \in \mathcal{T}_n$, there are exactly 2^n distinct automata $(\mathcal{T}, F)$.

Recall that the set $\mathcal{A}_n$ of deterministic, accessible and complete automata with n states is in bijection with the pairs $(\mathcal{T}, F)$ consisting of a deterministic, accessible and complete transition structure $\mathcal{T} \in \mathcal{T}_n$ with n states and a subset $F \subset \{1, \cdots, n\}$ of final states. Therefore, for the uniform distribution over the set $\mathcal{A}_n$, the average number of iterations of the main loop when Moore's algorithm is applied to an element of $\mathcal{A}_n$ is

$$\frac{1}{|\mathcal{A}_n|} \sum_{\mathcal{A} \in \mathcal{A}_n} \textsc{Moore}(\mathcal{A}) = \frac{1}{2^n |\mathcal{T}_n|} \sum_{\mathcal{T} \in \mathcal{T}_n} \sum_{F \subset \{1, \cdots, n\}} \textsc{Moore}(\mathcal{T}, F)$$

Using Proposition 3.4 we get

$$\frac{1}{|\mathcal{A}_n|} \sum_{\mathcal{A} \in \mathcal{A}_n} \textsc{Moore}(\mathcal{A}) \leq \frac{1}{|\mathcal{T}_n|} \sum_{\mathcal{T} \in \mathcal{T}_n} C \log n \leq C \log n.$$

Hence the average number of iterations is bounded by $C \log n$, and by Lemma 2.3, the average complexity of Moore's algorithm is upper bounded by $C_1 C n \log n$, concluding the proof. □

4. Tight bound for unary automata

In this section we prove that the bound $\mathcal{O}(n \log n)$ is optimal for the uniform distribution on unary automata with n states, that is, automata on a one-letter alphabet.

We shall use the following result on words, whose proof is given in detail in [8, p. 285]. For a word u on the binary alphabet $\{0, 1\}$, the *longest run of* 1 is the length of the longest consecutive block of 1's in u.

Proposition 4.1. *For any real number h and for the uniform distribution on binary words of length n, the probability that the longest run of 1 is smaller than $\lfloor \log_2 n + h \rfloor$ is equal to*

$$e^{-\alpha(n)2^{-h-1}} + \mathcal{O}\left(\frac{\log n}{\sqrt{n}}\right),$$

where the $\mathcal{O}$ is uniform on h, and $\alpha(n) = 2^{\log n - \lfloor \log n \rfloor}$.

Corollary 4.2. *For the uniform distribution on binary words of length n, the probability that the longest run of 1 is smaller than $\lfloor \frac{1}{2} \log_2 n \rfloor$ is smaller than $e^{-\sqrt{n}/2}$.*

Proof. Set $h = -\frac{1}{2} \log_2 n$ in Proposition 4.1 and use that for any integer n, $\alpha(n) \geq 1$. ∎

The shape of an accessible deterministic and complete automaton with n states on a one-letter alphabet $A = \{a\}$ is very specific. If we label the states using the depth-first order, then for all $q \in \{1, \cdots, n-1\}$ $q \cdot a = q + 1$. The state $n \cdot a$ entirely determines the transition structure of the automaton. Hence there are $n2^n$ distinct unary automata with n states. We shall also use the following result from [16]:

Proposition 4.3. *For the uniform distribution on unary automata with n states, the probability that an automaton is minimal is asymptotically equal to $\frac{1}{2}$.*

We can now prove the optimality of the $\mathcal{O}(n \log n)$ bound for unary automata:

Theorem 4.4. *For the uniform distribution on unary automata with n states, the average time complexity of Moore's state minimization algorithm is $\Theta(n \log n)$.*

Proof. From Theorem 3.1 this time complexity is $\mathcal{O}(n \log n)$. It remains to study the lower bound of the average time complexity of Moore's algorithm.

For any binary word u of size n, we denote by $F(u)$ the subset of $\{1, \cdots, n\}$ such that $i \in F(u)$ if and only if the i-th letter of u is 1. The map F is clearly a bijection between the binary words of length n and the subsets of $\{1, \cdots, n\}$. Therefore a unary automaton with n states is completely defined by a word u of length n, encoding the set of final states, and an integer $m \in \{1, \cdots, n\}$ corresponding to $n \cdot a$; we denote such an automaton by the pair $(u, m) \in \{0, 1\}^n \times \{1, \cdots, n\}$. Let ℓ be the integer defined by $\ell = \lfloor \frac{1}{2} \log_2 n \rfloor$. Let M_n be the set of minimal unary automata with n states, and S_n be the subset of M_n defined by

$$S_n = \{(u, m) \in M_n \mid \text{the longest run of 1 in } u \text{ is smaller than } \ell\}$$

As the number of element in S_n is smaller than the number of automata (u, m) whose longest run of 1 in u is smaller than ℓ, from Corollary 4.2, we have $|S_n| = o(n2^n)$. Let (u, m) be a minimal automaton in $M_n \setminus S_n$. The word u has a longest run of 1 greater or equal to ℓ. Let $p \in \{1, \cdots, n\}$ be the index of the beginning of such a longest run in u. The states p and $p + 1$ requires ℓ iterations in Moore's algorithm to be separated, as $p \cdot a^i$ and $(p + 1) \cdot a^i$ are both final for every $i \in \{0, \cdots, \ell - 2\}$. They must be separated

by the algorithm at some point since (u, m) is minimal. Hence $\textsc{Moore}((u, m)) \geq \ell$ for any $(u, m) \in M_n \setminus S_n$. Therefore

$$\begin{aligned}\frac{1}{n2^n} \sum_{(u,m)\in\{0,1\}^n\times\{1,\cdots,n\}} \textsc{Moore}((u, m)) &\geq \frac{1}{n2^n} \sum_{(u,m)\in M_n\setminus S_n} \textsc{Moore}((u, m)) \\ &\geq \frac{1}{n2^n}|M_n \setminus S_n|\ell \geq \frac{1}{n2^n}|M_n|\ell - \frac{1}{n2^n}|S_n|\ell \\ &\geq \frac{1}{2}\ell - o(\ell)\end{aligned}$$

The last inequality is obtained using Proposition 4.3, concluding the proof since by hypothesis $\ell = \lfloor \frac{1}{2} \log_2 n \rfloor$. ∎

5. Extensions

In this section we briefly present two extensions of Proposition 3.4 and Theorem 3.1.

5.1. Bernoulli distributions for the sets of final states

Let p be a fixed real number with $0 < p < 1$. Let $\mathcal{T}$ be a transition structure with n states. Consider the distribution on the sets of final states for $\mathcal{T}$ defined such that each state as a probability p of being final. The probability for a given subset F of $\{1, \cdots, n\}$ to be the set of final states is $\mathbb{P}(F) = p^{|F|}(1 - p)^{n-|F|}$.

A statement analogous to Proposition 3.4 still holds in this case. The proof is similar although a bit more technical, as Proposition 3.4 corresponds to the special case where $p = \frac{1}{2}$. Hence, for this distribution of sets of final states, the average complexity of Moore's algorithm is also $\mathcal{O}(n \log n)$.

5.2. Possibly incomplete automata

Now consider the uniform distribution on possibly incomplete deterministic automata with n states and assume that the first step of Moore's algorithm applied to an incomplete automaton consists in the completion of the automaton making use of a sink state. In this case Proposition 3.4 still holds. Indeed, Lemma 3.3 is still correct, even if the sets of final states F are the sets that do not contain the sink state. As a consequence, if a transition structure $\mathcal{T}$ is incomplete, the average complexity of Moore's algorithm for the uniform choice of set of final states of the completed transition structure, such that the sink state is not final, is in $\mathcal{O}((n + 1) \log(n + 1)) = \mathcal{O}(n \log n)$.

6. Open problem

We conjecture that for the uniform distribution on complete, accessible and deterministic automata with n states over a k-letter alphabet, with $k \geq 2$, the average time complexity of Moore's algorithm is in $\mathcal{O}(n \log \log n)$.

This conjecture comes from the following observations. First, Figure 2 seems to show a sub-logarithmic asymptotic number of iterations in Moore's algorithm. Second, if the automaton with n states is minimal, at least $\Omega(\log \log n)$ iterations are required to isolate every state: $\log n$ words are needed, and this can be achieved in the best case using all the words of length less than or equal to $\log \log n$. Moreover, in [2] we conjectured that a constant part of deterministic automata are minimal; if it is true, this would suggest that $\Omega(\log \log n)$ is a lower bound for the average complexity of Moore's algorithm. The conjecture above is that this lower bound is tight.

References

[1] M. Baclet, C. Pagetti, Around Hopcroft's Algorithm In *CIAA'2006* volume 4094 in Lect. Notes Comput. Sci., p. 114-125, Springer-Verlag, 2006.
[2] F. Bassino, C. Nicaud, Enumeration and random generation of accessible automata, *Theoret. Comput. Sci.*, 381, p. 86-104, 2007.
[3] F. Bassino, J. David, C. Nicaud, REGAL: a Library to randomly and exhaustively generate automata, in Jan Holub, Jan Zdarek, editors, *12th International Conference on Implementation and Application of Automata (CIAA'07)*, Vol. 4783 in Lect. Notes Comput. Sci., p. 303-305, Springer, 2007.
[4] F. Bassino, J. David, C. Nicaud, Random generation of possibly incomplete deterministic automata, in *Génération Aléatoire de Structures COMbinatoires (Gascom'08)*, p. 31- 40.
[5] J. Berstel, O. Carton, On the complexity of Hopcroft's state minimization algorithm, In *CIAA'2004*, Vol. 3317 of LNCS, p. 35-44, Springer-Verlag, 2004.
[6] J.A. Brzozowski, Canonical regular expressions and minimal state graphs for definite events, Proc. Symp. on. the Mathematical Theory of Automata, Vol. 12 of MRI Symposia Series, p. 529-561, Polytechnic Press, Polytechnic Institute of Brooklyn, New York, 1962.
[7] J.-M. Champarnaud, A. Khorsi, T. Paranthoen. *Split and Join for Minimizing: Brzozowski's algorithm* PSC'02 Proceedings, Prague Stringology Conference, Research report DC-2002-03, p. 96-104, 2002.
[8] P. Flajolet, R. Sedgewick, *Analytic combinatorics*, Cambridge University Press, 2009.
[9] D. Gries, Describing an Algorithm by Hopcroft, *Acta Inf.*, 2, p. 97-109, 1973.
[10] J.E. Hopcroft, J.D. Ullman *Introduction to Automata Theory, Languages and Computation.* Addison-Weisley Publishing Company, 1979.
[11] J.E. Hopcroft. *An $n \log n$ algorithm for minimizing the states in a finite automaton*, in The Theory of Machines and Computations, p 189-196, Academic Press, New York, 1971.
[12] T. Knuutila. Re-describing an algorithm by Hopcroft, *Theoret. Comput. Sci.*, 250, p. 333-363, 2001
[13] D. Korshunov. Enumeration of finite automata, *Problemy Kibernetiki*, 34, p. 5–82, 1978, In Russian.
[14] M. Lothaire. *Applied combinatorics on words*, Vol 105 of Encyclopedia of mathematics and its application. Cambridge University Press, 2005.
[15] E.F. Moore. *Gedanken experiments on sequential machines*, in C.E Shannon and J. McCarthy, Automata Studies, Princeton Univ. Press, p. 129-153, 1956.
[16] C. Nicaud. Average State Complexity of Operations on Unary Automata, In *MFCS 1999* volume 1672 in Lect. Notes Comput. Sci., p. 231 - 240, Springer-Verlag, 1999.
[17] B.W. Watson. A taxonomy of algorithms for constructing minimal acyclic deterministic finite automata *South African Computer Journal*, Vol. 27, p. 12-17, 2001.
[18] D. Wood *Theory of Computation.* John Wiley & Sons, 1987.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 135–146
www.stacs-conf.org

TESTING LINEAR-INVARIANT NON-LINEAR PROPERTIES

ARNAB BHATTACHARYYA [1] AND VICTOR CHEN [2] AND MADHU SUDAN [3] AND NING XIE [4]

[1] MIT CSAIL, Cambridge, MA 02139, USA
E-mail address: abhatt@csail.mit.edu

[2] MIT CSAIL, Cambridge, MA 02139, USA
E-mail address: victor@math.mit.edu

[3] MIT CSAIL, Cambridge, MA 02139, USA
E-mail address: madhu@csail.mit.edu

[4] MIT CSAIL, Cambridge, MA 02139, USA
E-mail address: ningxie@csail.mit.edu

Abstract. We consider the task of testing properties of Boolean functions that are invariant under linear transformations of the Boolean cube. Previous work in property testing, including the linearity test and the test for Reed-Muller codes, has mostly focused on such tasks for linear properties. The one exception is a test due to Green for "triangle freeness": A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ satisfies this property if $f(x), f(y), f(x+y)$ do not all equal 1, for any pair $x, y \in \mathbb{F}_2^n$.

Here we extend this test to a more systematic study of testing for linear-invariant non-linear properties. We consider properties that are described by a single forbidden pattern (and its linear transformations), i.e., a property is given by k points $v_1, \ldots, v_k \in \mathbb{F}_2^k$ and $f : \mathbb{F}_2^n \to \mathbb{F}_2$ satisfies the property that if for all linear maps $L : \mathbb{F}_2^k \to \mathbb{F}_2^n$ it is the case that $f(L(v_1)), \ldots, f(L(v_k))$ do not all equal 1. We show that this property is testable if the underlying matroid specified by $v_1, \ldots, v_k$ is a graphic matroid. This extends Green's result to an infinite class of new properties.

Our techniques extend those of Green and in particular we establish a link between the notion of "1-complexity linear systems" of Green and Tao, and graphic matroids, to derive the results.

1. Introduction

Property testing considers the task of testing, "super-efficiently", if a function $f : D \to R$ mapping a finite domain D to a finite range R essentially satisfies some desirable property. Letting $\{D \to R\}$ denote the set of all functions from D to R, a *property* is formally specified by a family $\mathcal{F} \subseteq \{D \to R\}$ of functions. A *tester* has oracle access to the function f and should accept with high probability if $f \in \mathcal{F}$ and reject (also with high probability) functions

Research supported in part by a DOE Computational Science Graduate Fellowship and NSF Awards 0514771, 0728645 and 0732334 .

Research supported in part by NSF Awards CCR-0514915 and 0829672.

Research supported in part by NSF Awards CCR-0514915 and 0829672.

Research supported in part by an Akamai Presidential Fellowship and NSF Awards 0514771, 0728645 and 0732334.

that are *far* from $\mathcal{F}$, while making very few queries to the oracle for f. Here, distance between functions $f, g : D \to R$, denoted $\delta(f, g)$, is simply the probability that $f(x) \neq g(x)$ when x is chosen uniformly at random from D and $\delta(f, \mathcal{F}) = \min_{g \in \mathcal{F}}\{\delta(f, g)\}$. We say f is δ-far from $\mathcal{F}$ if $\delta(f, \mathcal{F}) \geq \delta$ and δ-close otherwise. The central parameter associated with a tester is the number of oracle queries it makes to the function f being tested. In particular, a property is called *(locally) testable* if there is a tester with query complexity that is a constant depending only on the distance parameter δ. Property testing was initiated by the works of Blum, Luby and Rubinfeld [12] and Babai, Fortnow and Lund [9] and was formally defined by Rubinfeld and Sudan [25]. The systematic exploration of property testing was initiated by Goldreich, Goldwasser, and Ron [15] who expanded the scope of property testing to combinatorial and graph-theoretic properties (all previously considered properties were algebraic). In the subsequent years, a rich collection of properties have been shown to be testable [4, 5, 1, 13, 24, 3, 2, 21, 20] and many property tests have ended up playing a crucial role in constructions of probabilistically checkable proofs [8, 7, 11, 18, 27].

The rich collection of successes in property testing raises a natural question: Why are so many different properties turning out to be locally testable? Are there some broad "features" of properties that make them amenable to such tests? Our work is part of an attempt to answer such questions. Such questions are best understood by laying out broad (infinite) classes of properties (hopefully some of them are new) and showing them to be testable (or characterizing the testable properties within the class). In this paper we introduce a new such class of properties, and show that (1) they are locally testable, and (2) that they contain infinitely many new properties that were not previously known to be testable.

The properties, and our results: The broad scope of properties we are interested in are properties that view their domain D as a vector space and are invariant under linear transformations of the domain. Specifically, we consider the domain $D = \mathbb{F}_2^n$, the vector space of n-dimensional Boolean vectors, and the range $R = \mathbb{F}_2$. In this setting, a property $\mathcal{F}$ is said to be *linear-invariant* if for every $f \in \mathcal{F}$ and linear map $L : \mathbb{F}_2^n \to \mathbb{F}_2^n$ we have that $f \circ L \in \mathcal{F}$. Specific examples of linear-invariant properties that were previously studied (esp. in the Boolean setting) include that of linearity (studied by Blum et al. [12] and Bellare et al. [10]) and the property of being a "moderate-degree" polynomial (a.k.a. Reed-Muller codeword) studied by Alon et al. [2][1]. While the tests in the above mentioned works potentially used all features of the property being tested, Kaufman and Sudan [22] show that the testability can be attributed principally to the linear-invariance of the property. However their setting only considers *linear* properties, i.e., $\mathcal{F}$ itself is a vector space over $\mathbb{F}_2$ and this feature plays a key role in their results: It lends an algebraic flavor to all the properties being tested and plays a central role in their analysis.

We thus ask the question: Does linear-invariance lead to testability even when the property $\mathcal{F}$ is not linear? The one previous work in the literature that gives examples of non-linear linear-invariant properties is Green [16] where a test for the property of being "triangle-free" is described. A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is said to be *triangle-free* if for every $x, y \in \mathbb{F}_2^n$ it is the case that at least one of $f(x), f(y), f(x+y)$ does not equal 1. The property of being triangle-free is easily seen to be linear-invariant and yet not linear. Green [16] shows that the natural test for this property does indeed work correctly, though the analysis is

[1]In the literature, the term low-degree polynomial is typically used for polynomials whose degree is smaller than the field size. In the work of [2] the degrees considered are larger than the field size, but are best thought of as large constants. The phrase "moderate-degree" above describes this setting of parameters.

quite different from that of typical algebraic tests and is more reminiscent of graph-property testing. In particular, Green develops an algebraic regularity lemma to analyze this test. (We note that the example above is not the principal objective of Green's work, which is directed mostly at abelian groups D and R. The above example with $D = \mathbb{F}_2^n$ and $R = \mathbb{F}_2$ is used mainly as a motivating example.)

Motivated by the above example, we consider a broad class of properties that are linear-invariant and non-linear. A property in our class is given by k vectors $v_1, \ldots, v_k$ in the k-dimensional space $\mathbb{F}_2^k$. (Throughout this paper we think of k as a constant.) These k vectors uniformly specify a family $\mathcal{F} = \mathcal{F}_{n;v_1,\ldots,v_k}$ for every positive integer n, containing all functions that, for every linear map $L : \mathbb{F}_2^k \to \mathbb{F}_2^n$ take on the value 0 on at least one of the points $L(v_1), \ldots, L(v_k)$. (In the Appendix of the full version [14] we consider an even more generalized class of properties where the forbidden pattern of values for f is not 1^k but some other string and show a limited set of cases where we can test such properties.) To see that this extends the triangle-freeness property, note that triangle-freeness is just the special case with $k = 3$ and $v_1 = \langle 100\rangle$, $v_2 = \langle 010\rangle$, $v_3 = \langle 110\rangle$. Under different linear transforms, these three points get mapped to all the different triples of the form $x, y, x + y$ and so $\mathcal{F}_{n;v_1,v_2,v_3}$ equals the class of triangle-free functions.

Before giving a name to our class of functions, we make a quick observation. Note that the property specified by $v_1, \ldots, v_k$ is equivalent to the property specified by $T(v_1), \ldots, T(v_k)$ where T is a non-singular linear map from $\mathbb{F}_2^k \to \mathbb{F}_2^k$. Thus the property is effectively specified by the dependencies among $v_1, \ldots, v_k$ which are in turn captured by the matroid[2] underlying $v_1, \ldots, v_k$. This leads us to our nomenclature:

Definition 1.1. Given a (binary, linear) matroid $\mathcal{M}$ represented by vectors $v_1, \ldots, v_k \in \mathbb{F}_2^k$, the property of being *$\mathcal{M}$-free* is given by, for every positive integer n, the family

$$\mathcal{F}_{\mathcal{M}} = \{f : \mathbb{F}_2^n \to \mathbb{F}_2 | \forall \text{ linear } L : \mathbb{F}_2^k \to \mathbb{F}_2^n, \langle f(L(v_1)), \ldots, f(L(v_k))\rangle \neq 1^k\}.$$

The property of being $\mathcal{M}$-free has a natural k-local test associated with it: Pick a random linear map $L : \mathbb{F}_2^k \to \mathbb{F}_2^n$ and test that $\langle f(L(v_1)), \ldots, f(L(v_k))\rangle \neq 1^k\}$. Analyzing this test turns out to be non-trivial, and indeed we only manage to analyze this in special cases.

Recall that a matroid $\mathcal{M} = \{v_1, \ldots, v_k\}$, $v_i \in \mathbb{F}_2^k$, forms a *graphic matroid* if there exists a graph G on k edges with the edges being associated with the elements $v_1, \ldots, v_k$ such that a set $S \subset \{v_1, \ldots, v_k\}$ has a linear dependency if and only if the associated set of edges contains a cycle. In this paper, we require that the graph G be simple, that is, without any self-loops or parallel edges. Our main theorem shows that the property $\mathcal{F}$ associated with a graphic matroid $v_1, \ldots, v_k \in \mathbb{F}_2^k$ is testable.

Theorem 1.2. *For a graphic matroid $\mathcal{M}$, the property of being $\mathcal{M}$-free is locally testable. Specifically, let $\mathcal{M} = \{v_1, \ldots, v_k\}$ be a graphic matroid. Then, there exists a function $\tau : \mathbb{R}^+ \to \mathbb{R}^+$ and a k-query tester that accepts members of $\mathcal{M}$-free functions with probability one and rejects functions that are ϵ-far from being $\mathcal{M}$-free with probability at least $\tau(\epsilon)$.*

Our bound on τ is quite weak. We let $W(t)$ denote a tower of twos with height $\lceil t \rceil$. Our proof only guarantees that $\tau(\epsilon) \geq \frac{1}{W(poly(1/\epsilon))}$, a rather fast vanishing function. We do not know if such a weak bound is required for any property we consider.

[2]The definition of matroids may be found in, e.g., [30]. However a reader unfamiliar with this notion may just use the word matroid as a synonym for a finite collection of binary vectors, for the purposes of reading this paper.

We describe the techniques used to prove this theorem shortly (which shed light on why our bound on τ is so weak) but first comment on the implications of the theorem . First, note that for a graphic matroid it is more natural to associate the property with the underlying graph. We thus use the phrase G-free to denote the property of being $\mathcal{M}$-free where $\mathcal{M}$ is the graphic matroid of G. This terminology recovers the notion of being triangle-free, as in [16], and extends to cover the case of being k-cycle free (also considered in [16]). But it includes every other graph too!

Syntactically, Theorem 1.2 seems to include infinitely many new properties (other than being k-cycle free). However, this may not be true semantically. For instance the property of being triangle-free is essentially the same as being G-free for every G whose biconnected components are triangles. Indeed, prior to our work, it was not even explicitly noted whether being C_k-free is essentially different from being triangle-free. (By "essentially", we ask if there exist triangle-free functions that are *far* from being C_k-free.) It actually requires careful analysis to conclude that the family of properties being tested include (infinitely-many) new ones. Our second theorem addresses this point.

Theorem 1.3. *The class of G-free properties include infinitely many distinct ones. In particular:*

1. *For every odd k, if f is C_{k+2}-free, then it is also C_k-free. Conversely, there exist functions g that are C_k-free but far from being C_{k+2}-free.*
2. *If $k \leq \ell$ and f is K_k-free, then it is also K_ℓ-free. On the other hand, if $k \geq 3$ and $\ell \geq \binom{k}{2} + 2$ then there exists a function g that is K_ℓ-free but far from being K_k-free.*

Techniques: Our proof of Theorem 1.2 is based on Green [16]'s analysis of the triangle-free case. To analyze the triangle-free case, Green develops a "regularity" lemma for groups, which is analogous to Szemeri's regularity lemma for graphs. In our setting, Green's regularity lemma shows how, given any function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, one can find a subgroup H of $\mathbb{F}_2^n$ such that the restriction of f to almost all cosets of H is "regular", where "regularity" is defined based on the "Fourier coefficients" of f. (These notions are made precise in Section 3.1.)

This lemma continues to play a central role in our work as well, but we need to work further on this. In particular, a priori it is not clear how to use this lemma to analyze $\mathcal{M}$-freeness for *arbitrary* matroids $\mathcal{M}$. To extract a large feasible class of matroids we use a notion from a work of Green and Tao [17] of the complexity of a linear system (or matroids, as we refer to them). The "least complex" matroids have complexity 1, and we show that the regularity lemma can be applied to all matroids of complexity 1 to show that they are testable (see Section 3).

The notion of a 1-complex matroid is somewhat intricate, and a priori it may not even be clear that this introduces new testable properties. We show (in Section 4) that these properties actually capture all graphic matroids which is already promising. Yet this is not a definite proof of novelty, and so in Section 5 we investigate properties of graphic matroids and give some techniques to show that they are "essentially" different. Our proofs show that if two (binary) matroids are not "homomorphically" equivalent (in a sense that we define) then there is an essential difference between the properties represented by them.

Though our result on graphic matroids is derived from the notion of the complexity of systems of equations, the proof essentially boils down to "Fourier analysis on graphs". This notion had previously been considered and analyzed in the line of works investigating the amortized query complexity of PCPs [26, 19], where long-code tests based on graphs

were analyzed. One difference is that in their model, vertices correspond to labeled vectors whereas edges are labeled in our setting.

Though it's likely that one can show the testability of graphic matroids directly using similar techniques from [26] and [19], we remark that our technique gives a more inclusive viewpoint. First, non-graphic patterns are also shown to be testable. Second, we provide a framework toward an analytic proof of Green's conjecture.

Significance of problems/results: We now return to the motivation for studying $\mathcal{M}$-free properties. Our interest in these families is mathematical. We are interested in broad classes of properties that are testable; and invariance seems to be a central notion in explaining the testability of many interesting properties. Intuitively, it makes sense that the symmetries of a property could lead to testability, since this somehow suggests that the value of a function at any one point of the domain is no more important than its values at any other point. Furthermore this intuition is backed up in many special cases like graph-property testing (where the family is invariant under all permutations of the domain corresponding to relabeling the vertex names). Indeed this was what led Kaufman and Sudan [22] to examine this notion explicitly in the context of algebraic functions. They considered families that were linear-invariant and *linear*, and our work is motivated by the quest to see if the latter part is essential.

In contrast to other combinatorial settings, linear-invariance counts on a (quantitatively) very restricted collection of invariances. Indeed the set of linear transforms is only quasi-polynomially large in the domain (which may be contrasted with the exponentially large set of invariances that need to hold for graph-properties). So ability to test properties based on this feature is mathematically interesting and leads to the question: what kind of techniques are useful in these settings. Our work manages to highlight some of those (in particular, Green's regularity lemma).

Parallel Works: After completing our work, we learned from Asaf Shapira that, independently of us, $\mathcal{M}$-freeness for an arbitrary matroid $\mathcal{M}$ has been shown to be testable in Shapira's recent preprint [28]. This solves a question that we posed as open in an earlier version of this paper. His result is built on the work of Král', Serra, and Vena in [23], where an alternate proof of Green's cycle-freeness result is provided. Essentially the authors in [23] demonstrate a reduction from testing freeness of the cycle matroid in a function to testing freeness of the cycle subgraph in a graph, and then they apply regularity lemmas for graphs to analyze the number of cycles in a function far from being cycle-free. In this manner, the authors show that Theorem 1.2 holds as well. By extending this method and utilizing hypergraph regularity lemmas, Shapira [28] shows that arbitrary monotone matroid-freeness properties are testable.

We remark that our proofs are very different from [23] and [28], and in particular, our view on invariance leads us to develop techniques to show that syntactically different properties are indeed distinct.

Organization of this paper: In the following section (Section 2) we define a slightly broader class of properties that we can consider (including some non-monotone properties). We also define the notion of 1-complexity matroids which forms a central tool in our analysis of the tests. In Section 3 we show that for any 1-complexity matroid $\mathcal{M}$, $\mathcal{M}$-freeness is testable. In Section 4 we show that graphic matroids are 1-complexity matroids. Theorem 1.2 thus follows from the results of Section 3 and 4. In Section 5 we prove that there are infinitely many distinct properties among G-free properties. Due to space constraint we omit some proofs from this conference version. All the missing proofs as well as some additional results may be found in the full version of this paper [14].

2. Additional Definitions, Results, and Overview of Proofs

In this section, we describe some further results that we present in the paper and give an outline of proofs.

2.1. Extensions to Non-Monotone families

We first generalize Definition 1.1 to a wider collection of forbidden patterns.

Definition 2.1. Given $\Sigma \in \mathbb{F}_2^k$ and a binary matroid $\mathcal{M}$ represented by vectors $v_1, \ldots, v_k \in \mathbb{F}_2^k$, the property of being $(\mathcal{M}, \Sigma)$-*free* is given by, for every positive n, the family $\mathcal{F}_{(\mathcal{M},\Sigma)} = \{f : \mathbb{F}_2^n \to \mathbb{F}_2 | \forall \text{ linear } L : \mathbb{F}_2^k \to \mathbb{F}_2^n, \langle f(L(v_1)), \ldots, f(L(v_k))\rangle \neq \Sigma\}$.

If for some linear $L : \mathbb{F}_2^k \to \mathbb{F}_2^n$, $\langle f(L(v_1)), \ldots, f(L(v_k))\rangle = \Sigma$, then we say f *contains* $(\mathcal{M}, \Sigma)$ *at* L. Also for simplicity we suppress mention of Σ when $\Sigma = 1^k$.

Recall that a property $\mathcal{P} \subseteq \{D \to \{0,1\}\}$ is said to be *monotone* if $f \in \mathcal{P}$ and $g \prec f$ implies $g \in \mathcal{P}$, where $g \prec f$ means that $g(x) \leq f(x)$ for all $x \in D$.

Observation 2.2. For a binary matroid $\mathcal{M}$, $(\mathcal{M}, \Sigma)$-freeness is a monotone property if and only if $\Sigma = 1^k$.

In addition to our main results (Theorems 1.2 and 1.3) on monotone properties, we also obtain local testability results for a limited class of non-monotone properties.

Theorem 2.3. *Let C_k denote the cycle on k vertices and let Σ be an arbitrary element of $\mathbb{F}_2^k$. Then there exists a function $\tau : \mathbb{R}^+ \to \mathbb{R}^+$ and a k-query tester that accepts f in $\mathcal{F}_{(C_k,\Sigma)}$ with probability 1 and rejects f that are ϵ-far from $\mathcal{F}_{(C_k,\Sigma)}$ with probability at least $\tau(\epsilon)$.*

However, in strong contrast to Theorem 1.3, we show that unless Σ equals 0^k or 1^k, the class of (C_k, Σ)-freeness properties is not at all very rich semantically.

Theorem 2.4. *The class of properties $\{\mathcal{F}_{(C_k,\Sigma)} : k \geq 3, \Sigma \neq 0^k, \Sigma \neq 1^k\}$ is only finitely large.*

The goal of Theorem 2.3 is not to introduce new testable properties but rather to illustrate possible techniques for analyzing local tests that may lead to more classes of testable non-monotone properties.

2.2. Overview of Proofs

We now give an outline of the proofs of our main theorems (Theorems 1.2 and 1.3), and also the extensions (Theorems 2.3 and 2.4).

Our claim in Theorem 1.2, that graphic matroid freeness properties are locally testable, is based on analyzing the structure of dependencies among elements of a graphic matroid. To this end, we first recall the classification of linear forms due to Green and Tao in [17]. We require a minor reformulation of their definition since, for us, the structure of the linear constraints is described by elements of a matroid.

Definition 2.5. Given a binary matroid $\mathcal{M}$ represented by $v_1, \ldots, v_k \in \mathbb{F}_2^k$, we say that $\mathcal{M}$ has *complexity c at coordinate i* if we can partition $\{v_j\}_{j\in[k]\setminus\{i\}}$ into $c+1$ classes such that v_i is not in the span of any of the classes. We say that $\mathcal{M}$ has *complexity c* if c is the minimum such that $\mathcal{M}$ has complexity c at coordinate i for all $i \in [k]$.

The above definition makes sense because the span of a set of elements is not dependent on the specific basis chosen to represent the matroid. As a motivating example, consider the graphic matroid of C_k studied by Green in [16]. It can be represented by $v_1 = e_1, v_2 = e_2, \ldots, v_{k-1} = e_{k-1}$ and $v_k = e_1 + \cdots + e_{k-1}$. We see then that the graphic matroid of C_k has complexity 1 because for every $i < k$, the rest of the matroid elements can be partitioned into two sets $\{e_j\}_{j \neq i}$ and $\left\{\sum_{j \in [k]} e_j\right\}$ such that v_i is not contained in the span of either set, and for $i = k$, any nontrivial partition of the remaining elements ensures that v_k does not lie in the span of either partition. In Section 4, we extend this observation about C_k to all graphs.

Lemma 2.6. *For all graphs G, the graphic matroid of G has complexity* 1.

Green and Tao in [17] showed that if a matroid $\mathcal{M}$ has complexity c and if A is a subset of $\mathbb{F}_2^n$, then the number of linear maps $L : \mathbb{F}_2^k \to \mathbb{F}_2^n$ such that $L(v_i) \in A$ for all $i \in [k]$ is controlled by the $(c+1)$'th Gowers uniformity norm of A. Previously, Green proved in [16] an arithmetic regularity lemma, which essentially states that any set $A \subseteq \mathbb{F}_2^n$ can be partitioned into subsets of affine subspaces such that nearly every partition is nearly uniform with respect to linear tests. We show in Section 3 how to combine these two results to obtain the following:

Lemma 2.7. *Given any binary matroid $\mathcal{M}$ represented by $v_1, \ldots, v_k \in \mathbb{F}_2^k$, if $\mathcal{M}$ has complexity* 1*, then there exists a function $\tau : \mathbb{R}^+ \to \mathbb{R}^+$ and a k-query tester that accepts members of $\mathcal{F}_{\mathcal{M}}$ with probability* 1 *and rejects f that are ϵ-far from $\mathcal{F}_{\mathcal{M}}$ with probability at least $\tau(\epsilon)$.*

Theorem 1.2 directly follows from combining Lemma 2.6 and Lemma 2.7. In fact, Lemma 2.7 implies testability of all matroids that have complexity one, not only those that are graphic. In Section 4, we give examples of binary matroids that have complexity 1 and yet are provably not graphic.

Theorem 1.3 provides a proper hierarchy among the graphical properties. Moreover, the containments $\mathcal{P}_1 \subsetneq \mathcal{P}_2$ in this hierarchy are shown to be "statistically proper" in the sense that we demonstrate functions f that are ϵ-far from $\mathcal{P}_1$ but are in $\mathcal{P}_2$. The theorem implies the following hierarchy:

$$\cdots \subsetneq C_{k+2}\text{-free} \subsetneq C_k\text{-free} \subsetneq \cdots \subsetneq C_3\text{-free} = K_3\text{-free} \subsetneq \cdots \subsetneq K_k\text{-free} \subsetneq K_{\binom{k}{2}+2}\text{-free} \subsetneq \cdots$$

Thus, the class of properties $\mathcal{F}_G$ does indeed contain infinitely many more properties than the cycle freeness properties considered by Green in [16].

Both the hierarchy among the cyclic freeness properties and among the clique freeness properties are derived in Section 5 using a general technique. In order to show a statistically proper containment $\mathcal{M}_1$-free $\subsetneq \mathcal{M}_2$-free, we construct a function f that, by its definition, contains $\mathcal{M}_1$ at a large number of linear maps and so is far from being $\mathcal{M}_1$-free. On the other hand, the construction ensures that if f is also not $\mathcal{M}_2$-free, then there is a *matroid homomorphism* from $\mathcal{M}_2$ to $\mathcal{M}_1$. We define a matroid homomorphism from a binary matroid $\mathcal{M}_2$ to a binary matroid $\mathcal{M}_1$ to be a map from the ground set of $\mathcal{M}_2$ to the ground set of $\mathcal{M}_1$ which maps cycles to cycles. The separation between $\mathcal{M}_2$-freeness and $\mathcal{M}_1$-freeness is then obtained by proving that there do not exist any matroid homomorphisms from $\mathcal{M}_2$ to $\mathcal{M}_1$. This proof framework suffices for both the claims in Theorem 1.3 and is reminiscent of proof techniques involving graph homomorphisms in the area of graph property testing (see [6] for a survey).

Theorem 2.3 is the result of a more involved application of the regularity lemma. To deal with non-monotone properties, we employ a different "rounding" scheme inspired by the testability of non-monotone graph properties in [1]. Unlike Szemeri's regularity lemma, a "strong form" of the arithmetic regularity lemma is not known, so we restrict our attention to cyclic matroids and exploit the additive structure of the pattern. Theorem 2.4 is based on a characterization theorem that classifies (C_k, Σ)-freeness properties into 9 classes when $\Sigma \neq 0^k, 1^k$. Please see [14] for more details.

3. Freeness of Complexity 1 Matroids is Testable

In this section we prove Lemma 2.7. Before doing so, we fix our notation and provide a quick background on Fourier analysis. If H is a subgroup of G, the cosets of H are indicated by $g + H$, with g in G. Let $f_{g+H} : H \to \mathbb{F}_2$ denote f restricted to the coset $g + H$, defined by sending h to $f(g + h)$; that is, for every $h \in H, g \in G$, $f_{g+H}(h) := f(g + h)$. For $\sigma \in \mathbb{F}_2$, we define $\mu_\sigma(f_{g+H}) := \Pr_{h\in H}[f_{g+H}(h) = \sigma]$ to be the density of σ in f restricted to coset $g + H$.

3.1. Fourier Analysis and Green's Regularity Lemma

Definition 3.1 (Fourier transform)**.** If $f : \mathbb{F}_2^n \to \mathbb{F}_2$, then we define its Fourier transform $\widehat{f} : \mathbb{F}_2^n \to \mathbb{R}$ to be $\widehat{f}(\alpha) = \mathbb{E}_{x\in\mathbb{F}_2^n}[f(x)\chi_\alpha(x)]$, where $\chi_\alpha(x) = (-1)^{\sum_{i\in[n]} \alpha_i x_i}$. $\widehat{f}(\alpha)$ is called the Fourier coefficient of f at α, and the $\{\chi_\alpha\}_\alpha$ are the characters of $\mathbb{F}_2^n$.

It is easy to see that for $\alpha, \beta \in \mathbb{F}^n$, $\langle \chi_\alpha, \chi_\beta \rangle := \mathbb{E}_{x\in\mathbb{F}_2^n}[\chi_\alpha(x)\chi_\beta(x)]$ is 1 if $\alpha = \beta$ and 0 otherwise. So the characters form an orthonormal basis for $\mathbb{F}_2^n$, and we have the Fourier inversion formula $f(x) = \sum_{\alpha\in\mathbb{F}_2^n} \widehat{f}(\alpha)\chi_\alpha(x)$ and Parseval's Identity $\sum_{\alpha\in\mathbb{F}_2^n} \widehat{f}(\alpha)^2 = \mathbb{E}_x[f(x)^2] = \widehat{f}(0)$.

Next we turn to Green's arithmetic regularity lemma, the crux of the analysis of our local testing algorithm. Green's regularity lemma over $\mathbb{F}_2^n$ is a structural theorem for Boolean functions. It asserts that for every Boolean function, there is some decomposition of the Hamming cube into cosets, such that the function restricted to most of these cosets are uniform and pseudorandom with respect to the linear functions. An alternate and equivalent way is that no matter where we slice the Hamming cube by a hyperplane, the density of f on these cosets of the hyperplane is what we expect a random function looks like. Formally, we say that a function is uniform if all of its nonzero Fourier coefficients are small.

Definition 3.2 (Uniformity)**.** For every $0 < \epsilon < 1$, we say that a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is ϵ-uniform if for every $\alpha \neq 0 \in \mathbb{F}_2^n, |\hat{f}(\alpha)| \leq \epsilon$.

Recall that we let $W(t)$ denote a tower of twos with height $\lceil t \rceil$. To obtain a partition of the Hamming cube that satisfies the required uniformity requirement, the number of cosets in the partition may be rather large. More precisely,

Lemma 3.3 (Green's Regularity Lemma over $\mathbb{F}_2^n$)**.** *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$. Let $\epsilon \in (0, 1)$. Then there exists a subspace H of $G = \{0, 1\}^n$ of co-dimension at most $W(\epsilon^{-3})$, such that $\Pr_{g\in G}[f_{g+H}$ is ϵ-uniform$] \geq 1 - \epsilon$.*

3.2. Testability of Complexity 1 Matroid Freeness

The proposition below is proved in [17]. Collectively, statements capturing the phenomenon that expectation over certain forms are controlled by varying degrees of the Gowers norm are termed *generalized von-Neumann type Theorems* in the additive combinatorics literature. In particular, as we only require the degree 2 Gowers norm of a function, which is the sum of its Fourier coefficients raised to the fourth power, the following holds:

Proposition 3.4 ([17]). *Suppose a binary matroid $\mathcal{M} = \{v_1 \ldots, v_k\}$ has complexity 1 and let $f_1, \ldots, f_k : \mathbb{F}_2^n \to \mathbb{F}_2$. Then $\mathbb{E}_{L:\mathbb{F}_2^k \to \mathbb{F}_2^n}\left[\prod_{i=1}^k f_i(L(v_i))\right] \leq \min_{i\in[k]} \sum_{\alpha\in\mathbb{F}_2^n} \widehat{f_i}(\alpha)^4$.*

It is an easy deduction from Proposition 3.4 to see that if f is uniform, then the number of linear maps L where f has a $\mathcal{M}$-pattern is close to $\mathbb{E}[f]^m N^d$, where $N = 2^n$. Combining this observation with the regularity lemma, we prove Lemma 2.7.

Proof of Lemma 2.7. Consider a test that picks a linear map L uniformly at random from all linear maps from $\mathbb{F}_2^k \to \mathbb{F}_2^n$ and rejects iff for all $i \in [k]$, $f(L(v_i)) = 1$. Clearly the test has completeness one.

Now we analyze the soundness of this test. Suppose f is ϵ-far from being $\mathcal{M}$-free. We want to show that the test rejects with probability at least $\tau(\epsilon)$, such that $\tau(\epsilon) > 0$ whenever $\tau > 0$. Let $a(\epsilon)$ and $b(\epsilon)$ be two functions of ϵ that satisfy the constraint $a(\epsilon) + b(\epsilon) < \epsilon$, we shall specify these two functions at the end of the proof. We now apply Lemma 3.3 to f to obtain a subspace H of G of co-dimension at most $W(a(\epsilon)^{-3})$. Consequently, f restricted to all but at most $a(\epsilon)$ fraction of the cosets of H are $a(\epsilon)$-uniform. We define a reduced function $f^R : \mathbb{F}_2^n \to \mathbb{F}_2$ as follows.

For each $g \in G$, if f restricted to the coset $g + H$ is $a(\epsilon)$-uniform, then define

$$f^R_{g+H}(x) = \begin{cases} 0 & \text{if } \mu(f_{g+H}) \leq b(\epsilon) \\ f_{g+H} & \text{otherwise.} \end{cases}$$

Else, define $f^R_{g+H} = 0$.

Note that at most $a(\epsilon) + b(\epsilon)$ fraction of modification has been made to f to obtain f^R. Since f is ϵ-far from being $\mathcal{M}$-free, f^R has a $\mathcal{M}$-pattern at some linear map L. More precisely, for every $i \in [k]$, $f^R(L(v_i)) = 1$. Now consider the cosets $L(v_i) + H$. By our choice of rounding, we know that f restricted to each of these cosets is $a(\epsilon)$-uniform and at least $b(\epsilon)$ dense. We will count the number of linear maps $\phi : \mathbb{F}_2^k \to H$ such that f has a $\mathcal{M}$ pattern at $L + \phi$. Notice that the probability the test rejects is at least $2^{-k\cdot W(a(\epsilon)^{-3})} \Pr_{\phi:\mathbb{F}_2^k\to H}\left[\forall i, f_{L(v_i)+H}(\phi(v_i)) = 1\right]$.

To lower-bound this rejection probability, it suffices to show that the probability $\Pr_{\phi:\mathbb{F}_2^k\to H}\left[\forall i, f_{L(v_i)+H}(\phi(v_i)) = 1\right]$ is bounded below by at least some constant depending on ϵ. To this end, we rewrite this probability as $\mathbb{E}_{\phi:\mathbb{F}_2^k\to H}\left[\prod_{i\in[k]} f_i(\phi(v_i))\right]$, where $f_i = f_{L(v_i)+H}$. By replacing each function f_i by $\widehat{f_i}(0) + (f_i - \widehat{f_i}(0))$, it is easy to see that the above expression can be expanded into the sum of 2^k terms, one of which is $\prod_{i\in[k]} \widehat{f_i}(0)$, which is at least $b(\epsilon)^k$. For the other $2^k - 1$ terms, by applying Proposition 3.4 and using Parseval's Identity, each of these terms is bounded above by $a(\epsilon)^2$. So the expression is at least $b(\epsilon)^k - (2^k - 1)a(\epsilon)^2$. To finish the analysis, we need to specify $a(\epsilon), b(\epsilon)$ such that $b(\epsilon)^k - (2^k - 1)a(\epsilon)^2 > 0$ and $a(\epsilon) + b(\epsilon) < \epsilon$. Both are satisfied by setting $b(\epsilon) = \frac{\epsilon}{2}$, $a(\epsilon) = (\frac{\epsilon}{2})^k$. Thus, the rejection probability is at least $\tau(\epsilon) \geq 2^{-kW((\frac{2}{\epsilon})^{3k})} 2^{-k}(\epsilon^k - \epsilon^{2k})$, completing the proof. ∎

4. Graphic Matroids have Complexity 1

Here we prove that graphic matroids have complexity 1. While the proof is simple, we believe it sheds insight into the notion of complexity and shows that even the class of 1-complexity matroids is quite rich.

As we have seen earlier, Lemma 2.7 holds for any matroid of complexity 1. Hence, it is a natural question to ask whether there exist non-graphic matroids which have complexity 1. In the Appendix of the full version [14] we show that such matroids do exist. It is an open question to come up with a natural characterization of matroids having complexity 1.

5. Infinitely many Monotone Properties

In this section we prove Theorem 1.3, that there are infinitely many matroids for which the property of being $\mathcal{M}$-free are pairwise very different.

To do so we consider a pair of target matroids $\mathcal{M}_1$ and $\mathcal{M}_2$. Based on just the first matroid $\mathcal{M}_1$, we create a canonical function $f = f_{\mathcal{M}_1} : \mathbb{F}_2^n \to \mathbb{F}_2$. We show, using a simple analysis, that this canonical function is far from being $\mathcal{M}_1$ free. We then show that if this function has an instance of $\mathcal{M}_2$ inside, then there is a "homomorphism" (in a sense we define below) from $\mathcal{M}_2$ to $\mathcal{M}_1$. Finally we show two different ways in which one can rule out homomorphisms between pairs of graphic matroids; one based on the odd girth of the matroids, and the other based on the maximum degree of $\mathcal{M}_1$. Together these ideas lead to proofs of distinguishability of many different matroids.

Definition 5.1. Given a binary matroid $\mathcal{M}$ represented by vectors $v_1, \ldots, v_k \in \mathbb{F}_2^k$, and integer $n \geq k$, let the canonical function $f = f_{\mathcal{M}} : \mathbb{F}_2^n \to \mathbb{F}_2$ be given by $f(x, y) = 1$ if $x \in \{v_1, \ldots, v_k\}$ and 0 otherwise; where $x \in \mathbb{F}_2^k$ and $y \in \mathbb{F}_2^{n-k}$.

Claim 5.2. Let $\mathcal{M}$ be a binary matroid with $v_i \neq 0$ for all $i \in \{1, \ldots, k\}$. Then $f_{\mathcal{M}}$ is $\frac{1}{2^k}$-far from being $\mathcal{M}$-free.

We now introduce our notion of a "homomorphism" between binary matroids. (We stress that the phrase homomorphism is conjured up here and we are not aware of either this notion, or the phrase being used in the literature. We apologize for confusion if this phrase is used to mean something else.)

Definition 5.3. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be binary matroids given by $v_1, \ldots, v_k \in \mathbb{F}_2^k$ and $w_1, \ldots, w_\ell \in \mathbb{F}_2^\ell$. We say that $\mathcal{M}_2$ has a homomorphism to $\mathcal{M}_1$ if there is a map $\phi : \{w_1, \ldots, w_\ell\} \to \{v_1, \ldots, v_k\}$ such that for every set $T \subseteq [\ell]$ such that $\sum_{i \in T} w_i = 0$, it is the case that $\sum_{i \in T} \phi(w_i) = 0$.

For graphic matroids, the matroid-homomorphism from G to H is a map from the edges of G to the edges of H that ensures that cycles are mapped to even degree subgraphs of H.

Lemma 5.4. *If the canonical function $f_{\mathcal{M}_1}$ contains an instance of $\mathcal{M}_2$ somewhere, then $\mathcal{M}_2$ has a homomorphism to $\mathcal{M}_1$.*

The above lemma now motivates the search for matroids $\mathcal{M}_2$ that are not homomorphic to $\mathcal{M}_1$. Proving non-homomorphism in general may be hard, but we give a couple of settings where we can find simple proofs. Each addresses a different case of Theorem 1.3.

For a matroid $\mathcal{M}$, let its *odd girth*, denoted $\mathrm{og}(\mathcal{M})$, be the size of the smallest dependent set of odd cardinality, i.e. the size of the smallest odd set $T \subseteq [\ell]$ such that $\sum_{i \in T} w_i = 0$.

Lemma 5.5. *If $\mathcal{M}_2$ has a homomorphism to $\mathcal{M}_1$, then $\mathrm{og}(\mathcal{M}_2) \geq \mathrm{og}(\mathcal{M}_1)$.*

For graphic matroids constructed from the odd cycle graph C_k, we have that its odd girth is just k and so the above lemmas combine to give that C_k-freeness is distinguishable from C_{k+2}-freeness, and this suffices to prove Part (1) of Theorem 1.3.

However the odd girth criterion might suggest that G-freeness for any graph containing a triangle might be equivalent. Below we rule this possibility out.

Lemma 5.6. *Let $\mathcal{M}_1$ be the graphic matroid of the complete graph K_a on a vertices, and let $\mathcal{M}_2$ be the graphic matroid of K_b. Then, if $b \geq \binom{a}{2} + 2$, there is no homomorphism from $\mathcal{M}_2$ to $\mathcal{M}_1$.*

We are now ready to prove Theorem 1.3.

Proof of Theorem 1.3. First note that C_{k+2}-free functions are also C_k-free. Informally, suppose a function f has a k cycle at point $x_1, \ldots, x_k$, i.e., $f(x_i) = 1$ at these points and $\sum_i x_i = 0$. Then f has a $k+2$ cycle at the points $x_1, x_1, x_1, x_2, \ldots, x_k$. (This informal argument can obviously be converted to a formal one once we specify the graphic matroids corresponding to C_k and C_{k+2} formally.)

On the other hand, if we take $\mathcal{M}_1$ to be the graphic matroid corresponding to C_k and f to be the canonical function corresponding to $\mathcal{M}_1$, then by Claim 5.2 it is 2^{-k}-far from $\mathcal{M}_1$-free, and by Lemmas 5.4 and 5.5 it does not contain $\mathcal{M}_2$, the graphic matroid of C_{k+2}.

For the second part of the theorem, note that every property that is G-free is also H-free if G is a subgraph of H. Thus K_k-free is contained in K_ℓ free if $k \leq \ell$. The proper containment can now be shown as above, now using Claim 5.2 and Lemmas 5.4 and 5.6. ∎

6. Conclusions and Future Work

We introduced an infinite family of properties of Boolean functions and showed them to be testable. These properties were specified by a matroid $\mathcal{M}$ on k elements and a pattern $\Sigma \subseteq \{0,1\}^k$. However to capture the full range of linear-invariant non-linear properties that allow one-sided error local tests, we should also allow the conjunction of a constant number of constraints. We believe this could lead to a characterization of all linear-invariant non-linear properties that allow one-sided error local tests.

In a different direction, we feel that it would also be nice to develop richer techniques to show the distinguishability of syntactically different properties. For instance, even for the graphic case we don't have a good understanding of when two different graphs represent essentially the same properties, and when they are very different.

Acknowledgments

We are grateful to Kevin Matulef for suggesting this research direction. We thank Tali Kaufman and Swastik Kopparty for helpful discussions. We thank Asaf Shapira for drawing our attention to his preprint [28].

References

[1] Noga Alon, Eldar Fischer, Ilan Newman and Asaf Shapira. A combinatorial characterization of the testable graph properties: it's all about regularity. STOC'06:251–260, 2006.

[2] Noga Alon, Tali Kaufman, Michael Krivelevich, Simon Litsyn and Dana Ron, Testing low-degree polynomials over GF(2). Proceedings of Random 2003:188–199, 2003.

[3] Noga Alon, Michael Krivelevich, Ilan Newman and Mario Szegedy. Regular languages are testable with a constant number of queries. SIAM Journal on Computing, 30(6):1842-1862, 2000.

[4] Noga Alon and Asaf Shapira. Every monotone graph property is testable. STOC'05:128-137, 2005.

[5] Noga Alon and Asaf Shapira. A Characterization of the (natural) graph properties testable with one-sided error. FOCS'05:429-438, 2005.
[6] Noga Alon and Asaf Shapira. Homomorphisms in graph property testing - a survey. Electronic Colloquium on Computational Complexity (ECCC), Report TR05-085, 2005.
[7] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan and Mario Szegedy. Proof verification and the hardness of approximation problems. Journal of the ACM, 45(3):501–555, 1998.
[8] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. Journal of the ACM, 45(1):70–122, 1998.
[9] László Babai, Lance Fortnow and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. Computational Complexity, 1(1):3–40, 1991.
[10] Mihir Bellare, Don Coppersmith, Johan Håstad, Marcos A. Kiwi and Madhu Sudan. Linearity testing over characteristic two. IEEE Transactions on Information Theory, 42(6):1781–1795, 1996.
[11] Mihir Bellare, Oded Goldreich and Madhu Sudan. Free bits, PCPs, and nonapproximability—towards tight results. SIAM Journal on Computing, 27(3):804–915, 1998.
[12] Manuel Blum, Michael Luby and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. Journal of Computer and System Sciences, 47(3):549–595, 1993.
[13] Christian Borgs, Jennifer T. Chayes, László Lovász, Vera T. Sós, Balázs Szegedy and Katalin Vesztergombi. Graph limits and parameter testing. STOC'06:261-270, 2006.
[14] Arnab Bhattacharyya, Victor Chen, Madhu Sudan and Ning Xie. Testing linear-invariant non-linear properties. Electronic Colloquium on Computational Complexity (ECCC), Report TR08-088, 2008.
[15] Oded Goldreich, Shafi Goldwasser and Dana Ron. Property testing and its connection to learning and approximation. Journal of the ACM, 45(4):653–750, 1998.
[16] Ben Green. A Szemerédi-type regularity lemma in abelian groups, with applications. Geometric and Functional Analysis, 15(2):340–376, 2005.
[17] Ben Green and Terence Tao. Linear equations in primes. Annals of Mathematics, to appear.
[18] Johan Håstad. Some optimal inapproximability results. Journal of the ACM, 48(4):798–859, 2001.
[19] Johan Håstad and Avi Wigderson. Simple analysis of graph tests for linearity and PCP. Random Structures and Algorithms, 22(2):139–160, 2003.
[20] Charanjit S. Jutla, Anindya C. Patthak, Atri Rudra and David Zuckerman. Testing low-degree polynomials over prime fields. FOCS'04:423–432, 2004.
[21] Tali Kaufman and Dana Ron. Testing polynomials over general fields. FOCS'04:413–422, 2004.
[22] Tali Kaufman and Madhu Sudan. Algebraic property testing: the role of invariance. STOC'08: 403–412, 2008.
[23] Daniel Král', Oriol Serra and Lluís Vena. A combinatorial proof of the removal lemma for groups. arXiv:0804.4847, 2008.
[24] Michal Parnas, Dana Ron and Alex Samorodnitsky. Testing basic Boolean formulae. SIAM Journal on Discrete Mathematics, 16(1):20–46, 2003.
[25] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. SIAM Journal on Computing, 25(2):252-271, 1996.
[26] Alex Samorodnitsky and Luca Trevisan. A PCP characterization of NP with optimal amortized query complexity. STOC'00:191–199, 2000.
[27] Alex Samorodnitsky and Luca Trevisan. Gowers uniformity, influence of variables, and PCPs. STOC'06:11–20, 2006.
[28] Asaf Shapira. A proof of Green's conjecture regarding the removal properties of sets of linear equations. arXiv:0807.4901, 2008.
[29] William T. Tutte. Matroids and graphs. Transactions of the American Mathematical Society, 90:527–552, 1959.
[30] Dominic J.A. Welsh. Matroid Theory. Academic Press Inc., London, 1976.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 147–158
www.stacs-conf.org

KOLMOGOROV COMPLEXITY AND SOLOVAY FUNCTIONS

LAURENT BIENVENU [1] AND ROD DOWNEY [1]

[1] School of Mathematics, Statistics and Computer Science
Victoria University
P.O. Box 600
Wellington, New Zealand
E-mail address: {laurent.bienvenu,rod.downey}@mcs.vuw.ac.nz

Abstract. Solovay [19] proved that there exists a computable upper bound f of the prefix-free Kolmogorov complexity function K such that $f(x) = K(x)$ for infinitely many x. In this paper, we consider the class of computable functions f such that $K(x) \leq f(x)+O(1)$ for all x and $f(x) \leq K(x) + O(1)$ for infinitely many x, which we call Solovay functions. We show that Solovay functions present interesting connections with randomness notions such as Martin-Löf randomness and K-triviality.

1. Introduction

The plain and prefix-free Kolmogorov complexities (which we denote respectively by C and K) are both non-computable functions, but they do admit computable *upper bounds*. How good can these upper bounds be? That is, how close to C (resp. K) can a computable upper bound of C (resp. K) be? It can be easily proven that no computable upper bound of C can be close to C on all values, i.e. given any computable upper bound f of C, the ratio $f(x)/C(x)$ is not bounded. To see this, we use a variation of Berry's paradox: take a computable upper bound f of C, and define, for all $n \in \mathbb{N}$, x_n to be the smallest string x such that $f(x) \geq n$. Since f is computable, x_n can be computed from n, hence $C(x_n) \leq \log(n) + O(1)$. Thus, $f(x_n)/C(x_n) \geq n/(\log(n) + O(1))$ which proves the result. The exact same argument shows that no computable upper bound of K approximates K well on all values.

Therefore, one may ask the natural question: are there computable upper bounds for C or K that are good approximations on infinitely many values? The answer is trivially yes for C. Indeed, for most strings x, we have $C(x) = |x| + O(1)$ (see for example Downey and Hirschfeldt [7]), hence for some constant c, the function f defined by $f(x) = |x| + c$ is a computable upper bound of C, and $f(x) = C(x) + O(1)$ for infinitely many strings x. The case of K is less clear: indeed, the maximal prefix-free complexity of a string x of length n

1998 ACM Subject Classification: F.4.1.
Key words and phrases: Algorithmic randomness, Kolmogorov complexity, K-triviality.
The authors are supported by a grant from the Marsden fund of New Zealand.

(attained by most strings of that length) is $n + K(n) + O(1)$. And giving a good upper bound of this last expression already necessitates a good upper bound on K! Solovay [19] nonetheless managed to construct a computable upper bound f of K such that $f(x) = K(x)$ for infinitely many x. In this paper, we consider the class of computable functions f such that $K(n) \leq f(n) + O(1)$ for all n and $f(n) \leq K(n) + O(1)$ for infinitely many n, which we call Solovay functions.

Our first main result (Theorem 2.5) is that Solovay functions have a very simple characterization: they correspond to the computable functions f such that $\sum_x 2^{-f(x)}$ is finite and is a Martin-Löf random real.

Then, we discuss the role of Solovay functions in the characterization of randomness notions. In particular, we show (Theorem 3.4) that Solovay functions are particularly relevant to the Miller-Yu characterization of Martin-Löf random sequences via the plain Kolmogorov complexity of the initial segments. We prove along the way a theorem of independent interest (Theorem 3.5) showing that the Levin-Schnorr characterization of Martin-Löf randomness by prefix-free Kolmogorov complexity is very sharp, and derive several interesting consequences of this result.

Finally, we study two triviality notions that relate to computable upper bounds of prefix-free Kolmogorov complexity and Solovay functions. In the spirit of the Miller-Yu theorem, we obtain (Theorem 4.3) a characterization of K-triviality via computable upper bounds of K.

We assume that the reader is familiar with the field of algorithmic randomness. If not, one can consult Downey and Hirschfeldt [7] or Nies [18]. We denote by $2^{<\omega}$ and 2^ω the set of binary sequences (or "strings") and binary infinite sequences respectively. For a binary sequence x (finite or infinite), we denote by $x(i)$ the $(i+1)$-th bit of x, and by $x \upharpoonright i$ the string made of the first i bits of x (that is, $x \upharpoonright i = x(0)x(1)\dots x(i-1)$). The length of a string x is denoted by $|x|$. Throughout this paper, we identify $2^{<\omega}$ with $\mathbb{N}$, via the usual length-lexicographic bijection: $0 = \epsilon$ (ϵ being the empty string), $1 = 0$, $2 = 1$, $3 = 00$, $4 = 01$, $5 = 10\dots$. We also identify any element $r \in [0,1]$ to an element $\alpha \in 2^\omega$ such that $r = \sum_n \alpha(n) 2^{-n+1}$. If r is not dyadic then α is unique; if r is dyadic, there are two possible choices for $\alpha \in 2^\omega$ and which one we choose does not matter in this paper. We say that a real number is left-c.e. if it is the limit of a computable nondecreasing sequence of rational numbers. Given a nondrecreasing unbounded function $f : \mathbb{N} \to \mathbb{N}$, we denote by f^{-1} the function defined by $f^{-1}(k) = \min\{n \in \mathbb{N} \mid f(n) \geq k\}$. As we stated earlier, we denote by $C(x)$ and $K(x)$ the plain Kolmogorov complexity and prefix-free Kolmogorov of a string x. Since C and K are enumerable from above (i.e. their upper graph is a c.e. set), for a fixed enumeration, let $C_s(x)$ and $K_s(x)$ be the value of $C(x)$ and $K(x)$ at the s-th stage of the enumeration. In particular, this means that the function $(x, s) \mapsto K_s(x)$ is computable and, for any fixed x, $s \mapsto K_s(x)$ is nonincreasing and converges to $K(x)$ (and the same is true for C).

2. Computable upper bounds of Kolmogorov complexity

The class of computable upper bounds of Kolmogorov complexity has been studied in Bienvenu and Merkle [2] (in the setting of "decidable machines"), where they were used to give characterizations of a wide variety of randomness notions of randomness, such as

Martin-Löf randomness, Schnorr randomness, Kurtz randomness or computable dimension. Here, we are interested in the class of Solovay functions, which is a subclass of computable upper bounds of K (here and from now on, we use a slight abuse of terminology, calling "upper bound" of K a function f such that $K \leq f + O(1)$).

Let us first mention that computable upper bounds of K admit a very simple characterization.

Lemma 2.1. *Let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. The following are equivalent:*
(i) $K \leq f + O(1)$
(ii) The sum $\sum_{n\in\mathbb{N}} 2^{-f(n)}$ is finite.

Proof. $(i) \Rightarrow (ii)$ is trivial as $\sum_n 2^{-K(n)} \leq 1$. For $(ii) \Rightarrow (i)$, let c be such that $\sum_n 2^{-f(n)} \leq 2^c$. Using the Kraft-Chaitin theorem, we effectively construct a prefix-free c.e. set of strings $\{x_n \mid n \in \mathbb{N}\}$ with $|x_n| = f(n) + c$ for all n. Then, we define a (computable) function F by $F(x_n) = n$ for all n. Since F has prefix-free domain and is computable, we have $K(n) \leq |x_n| + O(1)$ for all n, hence $K(n) \leq f(n) + c + O(1)$. ∎

Definition 2.2. We denote by $\mathfrak{K}$ the class of computable functions such that $\sum_n 2^{-f(n)} < +\infty$ (or equivalently, the computable functions f such that $K \leq f + O(1)$).
We call *Solovay function* any function $f \in \mathfrak{K}$ such that $\liminf_{n\to+\infty} f(n) - K(n) < +\infty$ (or equivalently, any function $f \in \mathfrak{K}$ such that for some c, $f(n) \leq K(n) + c$ for infinitely many n).

Theorem 2.3 (Solovay [19])**.** *Solovay functions exist.*

Proof. Let us start by an observation. Given $x \in 2^{<\omega}$, and some p such that $\mathbb{U}(p) = x$, if we call t the computation time of $\mathbb{U}(p)$, we have

$$K\left(\langle x, p, t\rangle\right) \leq |p| + O(1)$$

(where $\langle ., ., .\rangle$ is a computable bijection from $2^{<\omega} \times 2^{<\omega} \times 2^{<\omega}$ to $2^{<\omega}$). Indeed, given p only, one can easily compute x and t. Suppose now that p is a *shortest* $\mathbb{U}$-program for x i.e. $\mathbb{U}(p) = x$ and $K(x) = |p|$. We then have:

$$|p| = K(x) \leq K\left(\langle x, p, t\rangle\right) \leq |p| + O(1)$$

Thus, let f be the function defined by:

$$f(\langle x, p, t\rangle) = \begin{cases} |p| \text{ if } \mathbb{U}(p) \text{ outputs } x \text{ in exactly } t \text{ steps of computation} \\ +\infty \text{ otherwise} \end{cases}$$

(here we use the value $+\infty$ for convenience, but any coarse upper bound of $K(\langle x, p, t\rangle)$, like $2|x| + 2|p| + 2\log t$, would do). By the above discussion, we have $K \leq f + O(1)$ and $f(\langle x, p, t\rangle) \leq K(\langle x, p, t\rangle) + O(1)$ for all triples (x, p, t) such that p is a shortest $\mathbb{U}$-program for x and $\mathbb{U}(p)$ outputs x in exactly t steps of computation. Thus, f is as desired. ∎

Remark 2.4. In fact, what Solovay actually proved is that there exists a computable function f such that $K \leq f$ and $K(n) = f(n)$ for infinitely many n. This can be easily deduced from Theorem 2.3. Indeed, given a computable function f such that $K \leq f + O(1)$ and $c = \liminf_{n\to+\infty} f(n) - K(n) < +\infty$, the (computable) function $f' = f - c$ is such that $f'(n) = K(n)$ for infinitely many n, and $f'(n) \geq K(n)$ for almost all n. Hence, up to modifying only finitely many values of f', we may assume that $f'(n) \geq K(n)$ for all n.

It turns out that, among the computable functions f such that the sum $\sum_n 2^{-f(n)}$ is finite, the Solovay functions are precisely those for which this sum is not only finite but also a Martin-Löf random real.

Theorem 2.5. *Let f be a computable function. The following are equivalent:*
(i) f is a Solovay function.
(ii) The sum $\sum_n 2^{-f(n)}$ is finite and is a Martin-Löf random real.

Proof. $(i) \Rightarrow (ii)$. If f is a Solovay function, we already know by definition that $\alpha = \sum_n 2^{-f(n)}$ is finite. Let us now prove that α is a Martin-Löf random real. Suppose it is not. Then for arbitrarily large c there exists k such that $K(\alpha \upharpoonright k) \leq k - c$ (this because of the Levin-Schnorr theorem, see next section). Given $\alpha \upharpoonright k$, one can effectively find some s such that

$$\sum_{n>s} 2^{-f(n)} \leq 2^{-k}$$

Thus, by a standard Kraft-Chaitin argument, one has $K(n|\alpha \upharpoonright k) \leq f(n) - k + O(1)$ for all $n > s$. Thus, for all $n > s$:

$$K(n) \leq f(n) + K(\alpha \upharpoonright k) - k + O(1) \leq f(n) + (k - c) - k - O(1) \leq f(n) - c - O(1)$$

And since c can be taken arbitrarily large, this shows that $\lim_{n\to+\infty} f(n) - K(n) = +\infty$ i.e. f is not a Solovay function.

$(ii) \Rightarrow (i)$. Suppose now for the sake of contradiction that f is not a Solovay function and that α is Martin-Löf random. We will prove that under these assumptions, the number $\Omega = \sum_n 2^{-K(n)}$ is not Martin-Löf random, which indeed is a contradiction (see for example Downey and Hirschfeldt [7]).

Since α is Martin-Löf random and is left-c.e., we can apply the Kučera-Slaman theorem [13]. This theorem states that given a Martin-Löf random left-c.e. real η, for any left-c.e. real ξ, there exists a constant d and a partial recursive function φ such that for every rational $q < \eta$, $\varphi(q)$ is defined and $\xi - \varphi(q) < 2^d(\eta - q)$. We will use this fact for $\eta = \alpha$ and $\xi = \Omega$ and also call d and φ the associated constant and partial recursive function.

Now, let c be a large integer (to be specified later). Suppose also that $\alpha \upharpoonright k$ is given for some k. Since $\alpha - (\alpha \upharpoonright k) < 2^{-k}$, by the Kučera-Slaman theorem:

$$\Omega - \varphi(\alpha \upharpoonright k) < 2^{-k+d}$$

Thus, from $\alpha \upharpoonright k$, one can effectively compute some $s(k)$ such that

$$\sum_{n>s(k)} 2^{-K(n)} \leq 2^{-k+d}$$

If k is large enough, then $s(k)$ is large enough and hence $n > s(k) \Rightarrow K(n) \leq f(n) - c - d$ (this because f is not a Solovay function). Thus, for all k large enough:

$$\sum_{n>s(k)} 2^{-f(n)} \leq 2^{-c-d} \sum_{n>s(k)} 2^{-K(n)} \leq 2^{-c-d} \cdot 2^{-k+d} \leq 2^{-k-c}$$

This tells us that for k large enough, knowing $\alpha \upharpoonright k$ suffices to compute $s(k)$ and then (by the above inequality) effectively compute an approximation of α by at most 2^{-k-c}. In other words, $\alpha \upharpoonright (k + c)$ can be computed from $\alpha \upharpoonright k$ and c. Therefore, for all k large enough:

$$K(\alpha \upharpoonright (k + c)) \leq K(\alpha \upharpoonright k, c) + O(1) \leq K(\alpha \upharpoonright k) + 2 \log c + O(1)$$

The constant d is fixed, and c can be taken arbitrarily large. Choose c such that the expression $2\log c + O(1)$ in the above inequality is smaller than $c/2$. Then, for all k large enough,

$$K(\alpha \upharpoonright (k+c)) \leq K(\alpha \upharpoonright k) + c/2$$

An easy induction then shows that $K(\alpha \upharpoonright k) \leq O(k/2)$, contradicting the fact that α is random. ∎

An interesting corollary of this theorem is that there are nondecreasing Solovay functions (which is not really obvious from the definition). To see that it is the case, it suffices to take a computable sequence $(r_n)_{n\in\mathbb{N}}$ of rational numbers such that every r_n is a negative power of 2, the r_n are nonincreasing and $\sum_n r_n$ is a Martin-Löf random number (it is very easy to see that such sequences exist). Then, take $f(n) = -\log(r_n)$ for all n. The function f is computable, nondecreasing and by Theorem 2.5 is a Solovay function.

3. Solovay functions and Martin-Löf randomness

One of the most fundamental theorems of algorithmic randomness is the Levin-Schnorr theorem, proven independently by Levin and Schnorr in the 1970's. It characterizes Martin-Löf random sequences by the prefix-free Kolmogorov complexity of their initial segments. More precisely, a sequence $\alpha \in 2^\omega$ is Martin-Löf random if and only if

$$K(\alpha \upharpoonright n) \geq n - O(1)$$

This theorem left open a fundamental question: is there a similar characterization of Martin-Löf randomness in terms of plain Kolmogorov complexity?

3.1. The Miller-Yu theorem

This question remained open for almost three decades. It was finally answered positively in a recent paper of Miller and Yu [16].

Theorem 3.1 (Miller and Yu[1] [16]). *Let $\alpha \in 2^\omega$. The following are equivalent:*
(i) α is Martin-Löf random.
(ii) $C(\alpha \upharpoonright n) \geq n - K(n) - O(1)$.
(iii) For all functions $f \in \mathfrak{K}$, $C(\alpha \upharpoonright n) \geq n - f(n) - O(1)$.

Remarkably, Miller and Yu showed that in the item (iii) above, the "for all f" part can be replaced by a *single* function:

Theorem 3.2 (Miller and Yu [16]). *There exists a function $g \in \mathfrak{K}$ such that for all $\alpha \in 2^\omega$:*

$$\alpha \text{ is Martin-Löf random } \Leftrightarrow\ C(\alpha \upharpoonright n) \geq n - g(n) - O(1) \tag{3.1}$$

Informally, the function $g \in \mathfrak{K}$ in this last proposition is a "good" upper bound of K, in the sense that it is close enough to K to make possible the replacement of K by g in the equivalence $(i) \Leftrightarrow (ii)$ of Theorem 3.1. This reminds us of the Solovay functions which are also "good" upper bounds of K in their own way. And indeed, the function g constructed by Miller and Yu to make the equivalence (3.1) true is a Solovay function. We will show that this is not a coincidence, as *all* functions g satisfying (3.1) are Solovay functions. But before that, we state a related theorem:

[1] Gács [10] proved the equivalence $(i) \Leftrightarrow (ii)$ of Theorem 3.1

Theorem 3.3 (Bienvenu and Merkle [2])**.** *A sequence α is Martin-Löf random if and only if for all $f \in \mathfrak{K}$, $f(\alpha \upharpoonright n) \geq n - O(1)$. Moreover, there exists a unique function $g \in \mathfrak{K}$ such that*

$$\alpha \text{ is Martin-Löf random } \Leftrightarrow \ g(\alpha \upharpoonright n) \geq n - O(1) \tag{3.2}$$

We will prove:

Theorem 3.4. *Any function g satisfying the equivalence (3.1) of Theorem 3.2 is a Solovay function. The same is true for any function g satisfying the equivalence (3.2) of Theorem 3.3.*

In order to prove this theorem, we show that in both characterizations of Martin-Löf randomness ($K(\alpha \upharpoonright n) \geq n - O(1)$ and $C(\alpha \upharpoonright n) \geq n - K(n) - O(1)$) the lower bound on complexity is very sharp, that is there is no "gap phenomenon".

3.2. A "no-gap" theorem for randomness

Chaitin [4] proved an alternative characterization of Martin-Löf randomness: $\alpha \in 2^\omega$ is Martin-Löf random if and only if $K(\alpha \upharpoonright n) - n$ tends to infinity. Together with the Levin-Schnorr characterization, this shows a dichotomy: given a sequence $\alpha \in 2^\omega$, either α is not Martin-Löf random, in which case $K(\alpha \upharpoonright n) - n$ takes arbitrarily large negative values, or α is Martin-Löf random, in which case $K(\alpha \upharpoonright n) - n$ tends to $+\infty$. This means for example that there is no sequence $\alpha \in 2^\omega$ such that $K(\alpha \upharpoonright n) = n + O(1)$. One may ask whether this dichotomy is due to a gap phenomenon, that is: is there a function h that tends to infinity, such that for every Martin-Löf random sequence α, $K(\alpha \upharpoonright n) \geq n + h(n) - O(1)$? Similarly, is there a function h' that tends to infinity such that for every sequence α, $K(\alpha \upharpoonright n) \geq n - h'(n) - O(1)$ implies that α is Martin-Löf random? We answer both these questions (and their plain complexity counterpart) negatively.

Theorem 3.5. *There exists no function $h : \mathbb{N} \to \mathbb{N}$ (computable or not) which tends to infinity and such that*

$$K(\alpha \upharpoonright n) \geq n - h(n) - O(1)$$

is a sufficient condition for α to be Martin-Löf random (in fact, not even for α to be Church stochastic).

Similarly, there is no function $h : \mathbb{N} \to \mathbb{N}$ which tends to infinity and such that

$$C(\alpha \upharpoonright n) \geq n - K(n) - h(n) - O(1)$$

is a sufficient condition for α to be Martin-Löf random (in fact, not even for α to be Church stochastic).

Proof. First, notice that since we want to prove this for *any* function that tends to infinity, we can restrict our attention to the nondecreasing ones. Indeed, if h is a function that tends to infinity, the function

$$\tilde{h}(n) = \min\{f(k) \mid k \geq n\}$$

also tends to infinity and $\tilde{h} \leq h$.

Now, assume we are in the simple case where the function h is nondecreasing and computable. A standard technique to get a non-random binary sequence β such that $K(\beta \upharpoonright n) \geq n - h(n) - O(1)$ is the following: take a Martin-Löf random sequence α,

and insert zeroes into α in positions $h^{-1}(0), h^{-1}(1), h^{-1}(2), \ldots$. It is easy to see that the resulting sequence β is not Martin-Löf random (indeed, not even Church stochastic), and that the Kolmogorov complexity of its initial segments is as desired. This approach was refined by Merkle et al. [15] where the authors used an insertion of zeroes on a co-c.e. set of positions in order to construct a left-c.e. sequence β that is not Mises-Wald-Church stochastic, but has initial segments of very high complexity.

Of course, the problem here is that the function h in the hypothesis may be non-computable, and in particular may grow slower than any computable nondecreasing function. In that case, the direct construction we just described does not necessarily work: indeed, inserting zeroes at a noncomputable set of positions may not affect the complexity of α. To overcome this problem, we invoke the Kučera-Gács theorem (see Kučera [12], Gács [11], or Merkle and Mihailovic [14]). This theorem states that any subset of $\mathbb{N}$ (or function from $\mathbb{N}$ to $\mathbb{N}$) is Turing-reducible to a Martin-Löf random sequence. Hence, instead of choosing *any* Martin-Löf sequence α, we pick one that computes the function h^{-1} and then insert zeroes into α at positions $h^{-1}(0), h^{-1}(1), \ldots$. Intuitively, the resulting sequence β should not be random, as the bits of α can be used to compute the places where the zeroes have been inserted. This intuition however is not quite correct, as inserting the zeroes may destroy the Turing reduction Φ from α to h^{-1}. In other words, looking at β, we may not be able to distinguish the bits of α from the inserted zeroes.

The trick to solve this last problem is to delay the insertion of the zeroes to "give enough time" to the reduction Φ to compute the positions of the inserted zeroes. More precisely, we insert the k-th zero in position $n_k = h^{-1}(k) + t(k)$ where $t(k)$ is the time needed by Φ to compute $h^{-1}(k)$ from α. This way, n_k is computable from $\alpha \upharpoonright n_k$ in time at most n_k. From this, it is not too hard to construct a computable selection rule that selects precisely the inserted zeroes, witnessing that β is not Church stochastic (hence not Martin-Löf random). Moreover, since the "insertion delay" only makes the inserted zeroes more sparse, we have $K(\beta \upharpoonright n) \geq n - h(n) - O(1)$. And similarly, since α is Martin-Löf random, we have by the Miller-Yu theorem: $C(\alpha \upharpoonright n) \geq n - K(n) - h(n) - O(1)$.

The formal details are as follows. Let h be a nondecreasing function. By the Kučera-Gács theorem, let α be a Martin-Löf random sequence and Φ be a Turing functional such that $\Phi^\alpha(n) = h^{-1}(n)$ for all n. Let $t(n)$ be the computation time of $\Phi^\alpha(n)$ (we can assume that t is a nondecreasing function). Let $\beta \in 2^\omega$ be the sequence obtained by inserting zeroes into α in positions $h^{-1}(n) + t(n)$. To show that β is not Church stochastic, we construct a (total) computable selection rule that filters the inserted zeroes from β. Let S be the selection rule that works as follows on a given sequence $\xi \in 2^\omega$. We proceed by induction; we call k_n the number of bits selected by S from $\xi \upharpoonright n$ and x_n the prefix $\xi \upharpoonright n$ of ξ from which these k_n bits are deleted (x_0 is thus the empty string, and $k_0 = 0$).

At stage $n+1$, having already read $\xi \upharpoonright n$, S computes $\Phi_n^{x_n}(k_n)$. If the computation halts after s steps, S checks whether $\Phi_n^{x_n}(k_n) + s$ returns n. If so, S selects the n-th bit of $\xi(n)$ of ξ and then sets $x_{n+1} = x_n$ and $k_{n+1} = k_n + 1$. Otherwise, S just reads the bit $\xi(n)$, and sets $x_{n+1} = x_n\xi(n)$ and $k_{n+1} = k_n$.

It is clear that S is a total computable selection rule. Now suppose that we run it on β. We argue that S selects exactly the zeroes that have been inserted into α to get β. We prove

this by induction. If S has already selected from β the first i inserted zeroes, then the next selected bit is the bit in position $n = \Phi^{x_n}(k_n) + s$ where $\Phi^{x_n}(k_n)$ is computed in s steps. But since the selected bits are exactly the zeroes that were inserted in α, we have $k_n = i$ and $x_n = \alpha \upharpoonright n - i$, and thus s is the computation time of $\Phi^{x_n}(k_n) = \Phi^{\alpha\upharpoonright n-i}(i)$, which we called $t(i)$. And by definition of Φ, $\Phi^{\alpha\upharpoonright n-i}(i) = h^{-1}(i)$. Therefore, $n = h^{-1}(i) + t(i)$, i.e. the selected bit was an inserted zero. This proves that S only selects bits that belong to the zeroes that were inserted into α. Conversely, we need to prove that all such bits are indeed selected by S. Let $i \in \mathbb{N}$. The $i+1$-th inserted zero is in position $n = h^{-1}(i) + t(i)$. At stage n, we have by the induction hypothesis $x_n = \alpha \upharpoonright n - i$ and $k_n = i$. Thus, $\Phi_n^{x_n}(k_n) = \Phi_{h^{-1}(i)+t(i)}^{\alpha\upharpoonright t(i)+h^{-1}(i)-i}(i)$, which has to halt because both quantities $t(i) + h^{-1}(i) - i$ and $h^{-1}(i)+t(i)$ are greater than $t(i)$, which is the computation time of $\Phi^{\alpha}(i)$. Thus the bit in position n is indeed selected. Therefore, S satisfies the desired properties, and witnesses the fact that β is not Church stochastic.

Finally, for all n, calling i the number of inserted zeroes in $\beta \upharpoonright n$, we easily see that $\beta \upharpoonright n$ and $\alpha \upharpoonright n - i$ can each be computed from the other one (by insertion or deletion of zeroes). Thus: $K(\beta \upharpoonright n) = K(\alpha \upharpoonright n - i) \geq n - i$ (since α is Martin-Löf random). And by definition of the positions where the zeroes are inserted, we have $n \geq h^{-1}(i-1) + t(i-1)$, hence $i \leq h(n) + O(1)$. Therefore:

$$K(\beta \upharpoonright n) \geq n - i \geq n - h(n) + O(1)$$

for all n. This completes the proof. ∎

As a consequence of the construction performed in this proof, we get the dual version of Theorem 3.5:

Proposition 3.6. *There exists no function $h : \mathbb{N} \to \mathbb{N}$ (computable or not) which tends to infinity and such that*

$$K(\alpha \upharpoonright n) \geq n + h(n) - O(1)$$

is a necessary condition for α to be Martin-Löf random.

Similarly, there is no function $h : \mathbb{N} \to \mathbb{N}$ which tends to infinity and such that

$$C(\alpha \upharpoonright n) \geq n - K(h) + h(n) - O(1)$$

is a necessary condition for α to be Martin-Löf random.

Proof. Suppose for the sake of contradiction that there exists a function h' which tends to infinity and such that $K(\alpha \upharpoonright n) \geq n + h'(n) - O(1)$ is a necessary condition for α to be Martin-Löf random. Once again, we can assume that h' is non-decreasing. Then, we perform the exact same construction as in the proof of Theorem 3.5 for a given function h. Then, at the end of proof, when evaluating the complexity of β, we have $K(\beta \upharpoonright n) = K(\alpha \upharpoonright n - i) + O(1)$, with $i \leq h(n) + O(1)$, and since α is Martin-Löf random, $K(\alpha \upharpoonright n - i) \geq (n - i) + h'(n - i) - O(1)$. It follows that

$$K(\beta \upharpoonright n) \geq n - h(n) + h'(n - h(n)) - O(1)$$

Thus, if we take h to be sufficiently slow growing (for example $h(n) = \log(h'(n))$), we have $K(\beta \upharpoonright n) \geq n - O(1)$ for all n. This is a contradiction since by the Levin-Schnorr theorem, this would imply that β is Martin-Löf random, which it is not by construction. The proof of the second part of the proposition is almost identical. ∎

Theorem 3.4 now easily follows:

Proof (of Theorem 3.4). Let g be a function satisfying the equivalence (3.1) of Theorem 3.2. Suppose that g is not a Solovay function. This means, by definition, that $h(n) = g(n) - K(n)$ tends to infinity. Then, we can rewrite the equivalence (3.1) as:

$$\alpha \text{ is Martin-Löf random } \Leftrightarrow\ C(\alpha \upharpoonright n) \geq n - K(n) - h(n) - O(1)$$

which contradicts Theorem 3.5. Similarly, if a function g satisfies the condition (3.2) of Theorem 3.3, and is such that $h(n) = g(n) - K(n)$ tends to infinity, then for all $\alpha \in 2^\omega$, α is Martin-Löf random if and only if $K(\alpha \upharpoonright n) \geq n - h(n)$, contradicting Theorem 3.5. ∎

The consequences of Theorem 3.5 go beyond its applications to Solovay functions. For example, it gives an alternative proof of the fact that Schnorr randomness does not imply Church stochasticity (a result originaly proven by Wang [20]). Indeed, it is rather well-known that if h tends to infinity slower than any computable nondecreasing function, then the condition $K(\alpha \upharpoonright n) \geq n - h(n) - O(1)$ is sufficient for α to be Schnorr random (see for example Bienvenu and Merkle [2]), whereas we just saw that it was not sufficient for α to be Church stochastic.

One can also adapt the proof of Theorem 3.5 to separate Church stochasticity from Schnorr randomness within the left-c.e. reals. Informally, this is done as follows. Take a left-c.e. Martin-Löf random sequence $\alpha \in 2^\omega$. Call $t(n)$ the settling time of $\alpha \upharpoonright n$, i.e. given a computable nondecreasing sequence $(q_s)_{s\in\mathbb{N}}$ that converges to α, $t(n)$ is the smallest s such that $|\alpha - q_s| < 2^{-n}$. It is easy to see that t is enumerable from below. Thus, the sequence $\beta \in 2^\omega$ which we obtain from α by inserting zeroes in positions $t(0) < t(1) < \dots$ is left-c.e. and for the same reason as above, is not Church stochastic. And the same kind of computation as above shows that $K(\beta \upharpoonright n) \geq n - t^{-1}(n) - O(1)$. Since it can easily be shown that t grows faster than any computable function, it follows that t^{-1} tends to infinity more slower than any nondecreasing unbounded computable function. Thus, β is not Church stochastic. This improves a result of Merkle et al. [15] (Theorem 26), who proved an equivalent fact for a weaker notion of stochasticity. For details on that result, see Bienvenu [1].

4. Solovay functions and triviality notions

A very successful line of research in algorithmic randomness over the last years concerns triviality and lowness notions. Informally, a sequence $\alpha \in 2^\omega$ is trivial if its Kolmogorov complexity is minimal or quasi-minimal, while a sequence α is low for randomness if it has little computation power, i.e. if relativizing the definition of random sequences to the oracle α does not change the class of random sequences. Perhaps the most important result in this direction was given by Nies [17]: a sequence $\alpha \in 2^\omega$ is low for Martin-Löf randomness (i.e. Martin-Löf randomness relativized to α coincides with standard Martin-Löf randomness) if and only if α is K-trivial (i.e. $K(\alpha \upharpoonright n) \leq K(n) + O(1)$). Other interesting notions of triviality have been studied, like Schnorr triviality, introduced by Downey et al. [6]: a sequence α is Schnorr trivial if for every prefix-free machine M whose domain has measure 1, there exists a machine M' whose domain also has measure 1, and such that $K_{M'}(\alpha \upharpoonright n) \leq K_M(n) + O(1)$. This notion was extensively studied by Franklin [8, 9].

In the same spirit, we can consider the class of sequences α such that for all computable upper bounds f of K, there exists a computable upper bound f' of K such that $f'(\alpha \upharpoonright n) \leq f(n) + O(1)$. However, because of the existence of Solovay functions, only computable sequences have this property.

Proposition 4.1. *Let $\alpha \in 2^\omega$. Suppose that for all $f \in \mathfrak{K}$, there exists $f' \in \mathfrak{K}$ such that*

$$f'(\alpha \upharpoonright n) \leq f(n) + O(1)$$

Then α is computable.

To prove this proposition, we need the following lemma:

Lemma 4.2 (Chaitin [3])**.** *For every $n, c \in \mathbb{N}$:*

$$\#\left\{w \in 2^{<\omega} \mid |w| = n \ \wedge \ K(w) \leq K(n) + c\right\} \leq 2^{c+O(1)}$$

where the $O(1)$ term does not depend on n or c.

Proof (of Proposition 4.1). Let $\alpha \in 2^\omega$ satisfy the hypothesis of the proposition. Let f be a Solovay function. By the assumption on α, there is a function $f' \in \mathfrak{K}$ and a constant c such that $f'(\alpha \upharpoonright n) \leq f(n) + c$ for all n. Let d be a constant such that $K \leq f' + d$. Since f is a Solovay function, there exists a constant e such that $f(n) \leq K(n) + e$ for infinitely many n. For any such n, we have:

$$\begin{aligned}
&\#\left\{w \in 2^{<\omega} \mid |w| = n \ \wedge \ f'(w) \leq f(n) + c\right\} \\
&\leq \#\left\{w \in 2^{<\omega} \mid |w| = n \ \wedge \ K(w) \leq K(n) + c + d + e\right\} \\
&\leq 2^{c+d+e+O(1)}
\end{aligned}$$

(the last inequality comes from Lemma 4.2). From this, we see that the Π^0_1 class

$$\{\xi \in 2^\omega \mid \forall n \ f'(\xi \upharpoonright n) \leq f(\xi \upharpoonright n) + c\}$$

to which α belongs, has only finitely many elements (at most $2^{c+d+e+O(1)}$), hence all these elements are computable. ∎

Another thing we can do is to study a weakened version of K-triviality: we consider the class of sequences α such that for any $f \in \mathfrak{K}$, $K(\alpha \upharpoonright n) \leq f(n) + O(1)$. As we shall now see, this is equivalent to K-triviality, hence we obtain an analogue of the Miller-Yu theorem for K-triviality.

Theorem 4.3. *Let $\alpha \in 2^\omega$. Then, α is K-trivial if and only if for all functions $f \in \mathfrak{K}$, $K(\alpha \upharpoonright n) \leq f(n) + O(1)$. Moreover, there exists a unique function $g \in \mathfrak{K}$ such that for all $\alpha \in 2^\omega$:*

$$\alpha \text{ is K-trivial} \Leftrightarrow K(\alpha \upharpoonright n) \leq g(n) + O(1) \tag{4.1}$$

Proof. By Lemma 2.1, it is obvious that any K-trivial α satisfies $K(\alpha \upharpoonright n) \leq f(n) + O(1)$ for all $f \in \mathfrak{K}$. Thus, all we have to do to prove this theorem is to construct a function g such that the implication "$\Leftarrow$" of equation (4.1) holds. In fact, we just take for g the function f constructed in the proof of Theorem 2.3.

Let then α be a sequence such that $K(\alpha \upharpoonright n) \leq g(n)+c$ for some constant c and all n. As usual, we prove that α is K-trivial by building a c.e. set L of pairs $(w_i, k_i)_{i \in \mathbb{N}}$ (with $w_i \in 2^{<\omega}$ and $k_i \in \mathbb{N}$) such that $\sum_i 2^{-k_i} < +\infty$ and for all n, some pair $(\alpha \upharpoonright n, K(n) + O(1))$ belongs

to L.

Let n be a fixed integer. We describe the strategy to enumerate strings of length n into L. We proceed by stages. At stage s, we look at the value of $K_s(n)$, and work under the assumption that $K_s(n) = K(n)$ (this assumption might turn out to be incorrect, we shall see below what to do when this happens). We then effectively find a $\mathbb{U}$-program p of length at most $K_s(n)$ such that $\mathbb{U}(p) = n$. By definition of g, if we call t the computation time of $\mathbb{U}(p)$, we have $g(\langle n, p, t\rangle) = |p| \leq K_s(n)$ (by definition of the function g), which, under the assumption $K_s(n) = K(n)$ implies $K(\langle n, p, t\rangle) = K(n)+O(1) = g(\langle n, p, t\rangle)+O(1)$. In other words, at every stage s, we can find a witness $m_s = \langle n, p, t\rangle$ such that $K(m_s) = g(m_s)+O(1)$, provided $K_s(n) = K(n)$.

Then, we enumerate all strings w of length m_s such that $K(w) \leq g(m_s) + c$, and for each such string we find, we put $(w \upharpoonright n, K_s(n))$ into L (without repetitions). Under the assumption $K_s(n) = K(n)$, we have $g(m_s) = K(m_s) + O(1)$ hence by Lemma 4.2, there are at most $d = 2^{c+O(1)}$ different strings w of length m_s such that $K(w) \leq g(m_s) + c$, hence at most d pairs of type $(w \upharpoonright n, K_s(n))$ enter L.

As we noted above, we might realize at some point that the assumption $K_s(n) = K(n)$ is incorrect, i.e. there might exist a stage $s' > s$ such that $K_{s'}(n) < K_s(n)$. In this case, we simply compute a new witness $m_{s'}$ and restart the strategy. However, the false assumption $g(m_s) = K(m_s) + O(1)$ may have caused us to enumerate many strings w of length m_s such that $K(w) \leq g(m_s) + c$ hence many pairs $(w \upharpoonright n, K_s(n))$ may enter L. We avoid this situation by only allowing d such pairs to enter L. Indeed, if more than d such pairs ask to enter L, we immediately know that the assumption $K_s(n) = K(n)$ is incorrect, hence we can stop acting and simply wait for a stage s' such that $K_{s'}(n) < K_s(n)$ and only then restart the strategy.

It remains to be verified that this strategy works, i.e. that the set L has the desired properties. For a fixed n, and any $k \geq K(n)$, by construction of L, there are at most d pairs of type $(w \upharpoonright n, k)$ in L. Thus, the total measure of the domain of L is at most

$$\sum_n \sum_{k \geq K(n)} d \cdot 2^{-k} = \sum_n d \cdot 2^{-K(n)+1} \leq 2d$$

hence is finite. Finally, for a given n, at some stage t we do have $K_t(n) = K(n)$. We then have $g(m_t) = K(m_t) + O(1)$ hence for *all* strings w of length m_t satisfying $K(w \upharpoonright m_t) \leq g(m_t) + c$, the pair $(wuhn, K_t(n))$ is enumerated into L (the restriction that at most d such pairs can enter L is not an actual restriction when $g(m_t) = K(m_t) + O(1)$). By definition of α, $K(\alpha \upharpoonright m_t) \leq g(m_t) + c$, hence by assumption $(\alpha \upharpoonright n, K_t(n)) = (\alpha \upharpoonright n, K(n))$ is enumerated into L. This completes the proof. ∎

We would like to end this paper with two questions.

Question 1. *Does* any *Solovay function g make the equivalence (3.1) of Miller-Yu's theorem true?*

Question 2. *Is any computable function g satisfying the equivalence (4.1) of Theorem 4.3 necessarily a Solovay function?*

Note that one cannot invoke a "no-gap" theorem to answer the second question, as it was noted by Csima and Montalbán [5] that there *is* a nondecreasing unbounded function h such that $K(\alpha \upharpoonright n) \leq K(n) + h(n) + O(1)$ implies that α is K-trivial.

Acknowledgement

The first author is grateful to Serge Grigorieff for very useful discussions on this work. The authors also thank the anonymous referees of this paper for helpful suggestions.

References

[1] Laurent Bienvenu. *Game-theoretic characterizations of randomness: unpredictability and stochasticity*. PhD thesis, Université de Provence, 2008.
[2] Laurent Bienvenu and Wolfgang Merkle. Reconciling data compression and Kolmogorov complexity. In *International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *Lecture Notes in Computer Science*, pages 643–654. Springer, 2007.
[3] Gregory Chaitin. Information-theoretical characterizations of recursive infinte strings. *Theoretical Computer Science*, 2:45–48, 1976.
[4] Gregory Chaitin. Incompleteness theorems for random reals. *Advances in Applied Mathematics*, 8:119–146, 1987.
[5] Barbara Csima and Antonio Montalbán. A minimal pair of K-degrees. *Proceedings of the American Mathematical Society*, 134:1499–1502, 2005.
[6] Rodney Downey, Evan Griffiths, and Geoffrey LaForte. On schnorr and computable randomness, martingales, and machines. *Mathematical Logic Quarterly*, 50(6):613–627, 2004.
[7] Rodney Downey and Denis Hirschfeldt. *Algorithmic randomness and complexity*. Springer, in preparation.
[8] Johanna Franklin. Hyperimmune-free degrees and Schnorr triviality. *Journal of Symbolic Logic*, 73:999–1008, 2008.
[9] Johanna Franklin. Schnorr trivial reals: a construction. *Archive for Mathematical Logic*, 46:665–678, 2008.
[10] Peter Gács. Exact expressions for some randomness tests. *Z. Math. Log. Grdl. M.*, 26:385–394, 1980.
[11] Peter Gács. Every set is reducible to a random one. *Information and Control*, 70:186–192, 1986.
[12] Antonin Kučera. Measure, Π^0_1 classes, and complete extensions of PA. *Lecture Notes in Mathematics*, 1141:245–259, 1985.
[13] Antonin Kučera and Ted Slaman. Randomness and recursive enumerability. *SIAM Journal on Computing*, 31:199–211, 2001.
[14] Wolfgang Merkle and Nenad Mihailovic. On the construction of effective random sets. *Journal of Symbolic Logic*, 69:862–878, 2004.
[15] Wolfgang Merkle, Joseph S. Miller, André Nies, Jan Reimann, and Frank Stephan. Kolmogorov-Loveland randomness and stochasticity. *Annals of Pure and Applied Logic*, 138(1-3):183–210, 2006.
[16] Joseph Miller and Liang Yu. On initial segment complexity and degrees of randomness. *Transaction of the American Mathematical Society*, 360(6):3193–3210, 2008.
[17] André Nies. Lowness properties and randomness. *Advances in Mathematics*, 197(1):274–305, 2005.
[18] André Nies. *Computability and randomness*. Oxford University Press, To appear.
[19] Robert Solovay. Draft of a paper (or series of papers) on Chaitin's work. Unpublished notes, 215 pages, 1975.
[20] Yongge Wang. A separation of two randomness concepts. *Information Processing Letters*, 69(3):115–118, 1999.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 159–170
www.stacs-conf.org

WEAK MSO WITH THE UNBOUNDING QUANTIFIER

MIKOŁAJ BOJAŃCZYK

University of Warsaw
E-mail address: bojan@mimuw.edu.pl
URL: www.mimuw.edu.pl/∼bojan

Abstract. A new class of languages of infinite words is introduced, called the *max-regular languages*, extending the class of ω-regular languages. The class has two equivalent descriptions: in terms of automata (a type of deterministic counter automaton), and in terms of logic (weak monadic second-order logic with a bounding quantifier). Effective translations between the logic and automata are given.

1. Introduction

This paper introduces a new class of languages of infinite words, which are called *max-regular languages*, and include all ω-regular languages. Max-regular languages can be described in terms of automata, and also in terms of a logic. A typical language in the class is the property "the distance between consecutive b's is unbounded", i.e. the language

$$L = \{a^{n_1}ba^{n_2}ba^{n_3}\ldots : \forall m\ \exists i\ n_i > m\}\ . \tag{1.1}$$

A practical motivation can be given for considering properties that speak of bounded distance; e.g. a formula of the logic in this paper could specify that a system responds to requests with bounded delay. We will begin, however, with a more fundamental motivation, which is the question: what is a regular language of infinite words?

There is little doubt as to what is a regular language of finite words. For instance, the requirement that the Myhill-Nerode equivalence relation has finitely many equivalence classes uniquely determines which languages of finite words should be regular. Other notions, such as finite semigroups, or monadic-second order logic also point to the same class.

For infinite words, however, there is more doubt. Of course, the class of ω-regular languages has much to justify calling it regular, but some doubts remain as to its uniqueness. Consider, for instance, the language L mentioned above, or the set K of ultimately periodic words, i.e. words of the form wv^ω, say over alphabet a, b. None of these languages are ω-regular. However, under the commonly accepted definition of Myhill-Nerode equivalence for infinite words, given by Arnold in [2], both languages have exactly one equivalence class.

Should these languages be called regular? If yes, what is the appropriate notion of regularity? In this paper we propose a notion of regular languages, which are called *max-regular*

Key words and phrases: automata, monadic second-order logic.
Author supported by Polish government grant no. N206 008 32/0810.

languages, that captures the language L, but not the language K. This new notion has many properties that one would wish from regular languages. The class is (effectively) closed under boolean operations, including negation. There is a finite index Myhill-Nerode relation, and equivalence classes are regular languages of finite words. There is an automaton model, there is a logical description, and translations between the two are effective. Emptiness is decidable. Membership is decidable (although since we deal with infinite words, the membership test is for certain finitely presented inputs, such as ultimately periodic words).

So, what is this new class? One definition is in terms of logic. The max-regular languages are the ones that can be defined by formulas of weak monadic second-order logic extended with the unbounding quantifier. The term "weak" means that only quantification over finite sets is allowed. The unbounding quantifier $UX.\varphi(X)$ was introduced[1] in [3], it says that the size of sets X satisfying $\varphi(X)$ is unbounded, i.e.

$$UX.\varphi(X) = \bigwedge_{n\in\mathbb{N}} \exists X \ \big(\ \varphi(X) \quad \wedge \quad n \leq |X| < \infty \ \big). \tag{1.2}$$

Monadic second-order logic with the unbounding quantifier for infinite trees was studied in [3], where an emptiness procedure was presented for formulas with restricted quantification patterns. This study was continued in [4], where the models where restricted from infinite trees to infinite words, but the quantification patterns considered were more relaxed. However, no decision procedure was given in [4] for full monadic second-order logic with the unbounding quantifier, and the expressive power of the logic seemed to be far too strong for the techniques used (no undecidability results are known, though).

The basic idea in this paper is to restrict the set quantification to finite sets (i.e. weak quantification), while keeping the unbounding quantifier. It turns out that with this restriction, lots of the problems encountered in [4] are avoided, and the resulting class is surprisingly robust. Note that for infinite words and without unbounding quantification, weak monadic second-order logic has the same expressive power as full monadic second-order logic; this is no longer true when the unbounding quantifier is allowed (we prove this using topological techniques).

The main contribution of this paper is Theorem 3.2, which shows that weak monadic second-order logic with the unbounding quantifier has the same expressive power as deterministic max-automata. A max-automaton is a finite automaton equipped with counters, which store natural numbers. The important thing is that the counters are not read during the run (and therefore do not influence the control of the automaton), which avoids the usual undecidability problems of counter machines. The counters are only used in the acceptance condition, which requires some counter values to be bounded, and some to be unbounded.

To the best of the authors knowledge, quantifiers similar to the unbounding quantifier have only been considered in [3, 4]. On the other hand, the idea to use automata with quantitative acceptance conditions, has a long history, going back to weighted automata of Schützenberger [11] (see [7] for a recent paper on weighted automata and related logics).

The max-automata used in this paper are closely related to an automaton model that has been variously called a *distance desert automaton* in [10], a *BS-automaton* in [4], or an *R-automaton* in [1]. One important application, see [10], of these automata is that they can

[1]The quantifier introduced in [3] was actually the negation of U, saying that the size is bounded.

be used to solve the famous star-height problem[2], providing simpler techniques and better complexities than in the famous result of Hashiguchi [8]. (The reduction from the star-height problem is not to emptiness of the automata, but to something called *limitedness*.) Other problems that can be tackled using this type of automata include the star-height of tree languages [5] or the Mostowski index of ω-regular languages [6].

2. The automaton

We begin our presentation with the automaton model.

A *max-automaton* has a finite set of states Q and a finite set of counters Γ. It also has a finite set of transitions. Each transition reads an input letter, changes the state, and does a finite sequence of counter operations. The counter operations are:

$c := c + 1$.	Increment counter c.
$c := 0$.	Reset counter c.
$output(c)$.	Output the value of counter c.
$c := max(c, d)$.	Store in counter c the maximal value of counters c, d.

A max-automaton is run on an infinite word $w \in \Sigma^\omega$. A run is an infinite sequence of transitions, with the usual requirement on consistency with the letters in the input word. Fix a run ρ. With each counter $c \in C$, we associate the sequence counter values $\rho_c \in \mathbb{N}^* \cup \mathbb{N}^\omega$ that have been output by the instruction $output(c)$. These outputs are used by the accepting condition, which is a boolean combination of clauses: "the sequence ρ_c is bounded".

Note that with this acceptance condition, it is only the set of values in ρ_c that matters, and not their order or multiplicity. This is unlike the parity condition (where multiplicity is important), or the S-condition of [4], where the sequence ρ_c is required to tend to infinity.

The toolkit of counter operations could be modified without affecting the expressive power of max-automata. For instance, we could have an operation $c := d$, which is equivalent to $c := 0$ followed by $c := max(c, d)$. On the other hand, the output instruction can be removed (in this case, ρ_c would contain all values of the counter during the run). The output operation can be simulated by the others as follows: for every counter c, we add a new output counter c', which is never incremented. Instead of doing $output(c)$, we do $c' := c$. This way, the counter c' gets only the values that were output on the original counter c.

Theorem 2.1. *Emptiness is decidable for max-automata.*

Proof. The difficulty in the proof is dealing with the max operation.

We will reduce the problem to a result from [4]. A direct and elementary proof can also be given. A *U-automaton* is a max-automaton that does not use the max operation, and where the acceptance condition is a *positive* boolean combination of clauses "counter c is unbounded".

Let $\mathcal{A}$ be a max-automaton that we want to test for emptiness. As is often the case, we will be searching not for an input word accepted by $\mathcal{A}$, but for an accepting run of $\mathcal{A}$ (which is also an infinite word). Fix a single clause in the accepting condition, e.g. "counter c is unbounded". Below, we will show that the set of runs which satisfy this clause can be recognized by a nondeterministic U-automaton. In particular, the set of accepting runs of

[2]This is the question of calculating the least number of nested stars in a regular expression (without negation) that defines a regular language $L \subseteq \Sigma^*$.

$\mathcal{A}$ is a boolean combination of languages accepted by U-automata. The result then follows from [4], where emptiness is shown decidable for boolean combinations of nondeterministic U-automata[3].

Before we define the U-automaton that tests if counter c is unbounded, we introduce some auxiliary definitions. Let c, d be counters of the automaton $\mathcal{A}$. Below we define what it means for a finite sequence of counter operations ρ to transfer c to d, possibly with an increment. (Formally, we are defining two ternary relations: $T(\rho, c, d)$, for transfers, and $TI(\rho, c, d)$, for transfers with an increment.) The idea is that after executing the operations ρ, the value of counter d is at least as big as the value of counter c before executing ρ. The definition of transfers is by induction on the length of ρ:

- Every counter is transferred to itself by the empty sequence of operations, as well as the operations $c := c + 1$ and $output(c)$. Furthermore, $c := c + 1$ also transfers c to itself with an increment.
- The operation $c := 0$ transfers every counter to itself, except c.
- The operation $c = \max(c, d)$ transfers every counter to itself, and also d to c.
- If a sequence of operations ρ_1 transfers c to e, and a sequence of operations ρ_2 transfers e to d, then their concatenation $\rho_1\rho_2$ transfers c to d. If either of the transfers in ρ_1 or ρ_2 does an increment, then so does the transfer in $\rho_1\rho_2$.

Note that the transfer relation is regular in the following sense: for any counters c and d, the set of words ρ that transfer counter c to d is a regular language of finite words, likewise for transfers with an increment.

Let c be a counter. A finite sequence of positions $x_1 < \cdots < x_n$ in a run of $\mathcal{A}$ is called a *c-loop* if for any $i < n$, counter c is transferred to itself with an increment by the subrun between positions x_i to x_{i+1}. For a counter d, a *d-trace* is a sequence of positions $x_1 < \cdots < x_n < y$ such that for some counter c, the positions $x_1 < \cdots < x_n$ are a c-loop, and counter c is transferred to d by the subrun between positions x_n and y.

Equipped with these definitions, we are ready to define a (nondeterministic) U-automaton that tests if counter c is unbounded in an input run. The U-automaton has only one counter, and it accepts if unbounded values are output to this counter. A run of this automaton (which inputs a run of the automaton $\mathcal{A}$) proceeds as follows. It uses nondeterminism to guess a d-trace $x_1 < \cdots < x_n < y$, and it increments its counter at each of the positions x_i. Once it sees position y, it outputs the counter value (which is n), and resets the counter. It then finds another d-trace, and again outputs its length, and so on. It is not difficult to verify the correctness of this construction. ∎

In this paper, we will be mainly interested in deterministic max-automata.

3. The logic

We consider an extension of weak monadic second-order logic, called *weak unbounding logic*. Recall that weak monadic second-order logic is an extension of first-order logic that allows quantification over finite sets (the restriction to finite sets is the reason for the name "weak"). In weak unbounding logic, we further add the *unbounding quantifier* UX, as defined in (1.2).

[3]The result in [4] is for S-automata, which are more powerful than U-automata. It is shown that a boolean combination of S-automata is equivalent to a BS-automaton, which has decidable emptiness.

Example 3.1. Consider the set L from (1.1). This language is not regular, but defined by the following formula of weak unbounding logic:

$$UX\ \forall x \leq y \leq z \quad x, z \in X\ \Rightarrow\ a(y) \wedge y \in X$$

The main result of this paper is that the logic and automata coincide, i.e.

Theorem 3.2. *Weak unbounding logic defines exactly the same languages as deterministic max-automata.*

The more difficult direction in Theorem 3.2 is presented in Section 4. The easier direction, where an automaton is simulated by the logic, can be shown by combining standard techniques with the concepts from the proof of Theorem 2.1. The key idea is that a formula of weak unbounding logic can test if a set of positions $\{x_1 < \cdots < x_n < y\}$ forms a d-trace. It is important that the automata are deterministic, which allows a formula of weak logic to uniquely decode the run that corresponds to the input word.

The formulas that are sufficient to simulate a deterministic max-automaton are of a special type, which gives a normal form for weak unbounding logic:

Proposition 3.3. *Each formula of weak unbounding logic is equivalent to a boolean combination of formulas $UX\varphi(X)$, where $\varphi(X)$ does not use the unbounding quantifier.*

Proof. By translating a formula into an automaton and then back into a formula. ∎

4. Weak bounding logic is captured by deterministic max-automata

We now turn to the more difficult part of Theorem 3.2, namely showing that for every formula of weak unbounding logic there is an equivalent deterministic max-automaton.

The proof is by induction on the size of the formula. To simplify the proof, we use the usual technique of removing first-order quantification, as in [13]. That is, first-order quantification is replaced by three new predicates, all of which can be recognized by the deterministic max-automata: "set X has one element", "set X is included in set Y" and "all elements of set X are before all elements of set Y". Together with weak second-order quantification, these new three predicates can be used to simulate first-order quantification, so the logic is the same. However, since we have removed first-order quantification, in the translation to automata we only have to deal with quantification over finite sets (weak second-order quantification) and the new quantifier.

For purposes of the induction, we generalize the statement to formulas with free variables. What is the word language corresponding to a formula $\varphi(X_1, \ldots, X_n)$? This language contains words annotated with valuations for the free set variables. We use the usual encoding, where the label of a word position $x \in \mathbb{N}$ is extended with a bit vector in $\{0,1\}^n$ that says which of the sets $X_1, \ldots, X_n$ contain position x. More formally, for sets of word positions $X_1, \ldots, X_n \subseteq \mathbb{N}$ and an infinite word $w \in \Sigma^\omega$, we define the word

$$w[X_1, \ldots, X_n]\ \in\ (\Sigma \times \{0,1\}^n)^\omega$$

as follows. On position x, the new word has a tuple $(a, b_1, \ldots, b_n)$, with a the label of the x-th position of the original word w, and the value of bit b_i being 1 if and only if position x belongs to the set X_i, for $i = 1, \ldots, n$. With this notation, we can define the set of words satisfying a formula $\varphi(X_1, \ldots, X_n)$ to be

$$L_\varphi = \{w[X_1, \ldots, X_n] : w, X_1, \ldots, X_n \models \varphi\}\ .$$

Equipped with the above definition, we can use induction to show that the logic is captured by automata, as stated in the proposition below. This result is the main ingredient in the proof of Theorem 3.2.

Proposition 4.1. *For every formula φ of weak unbounding logic, the set L_φ is recognized by a deterministic max-automaton.*

The proof is by induction on the size of the formula φ. The induction base, which corresponds to the predicates "set X has one element", "set X is included in set Y" and "all elements of set X are before all elements of set Y" is easy, since all of these are ω-regular languages, and we have:

Lemma 4.2. *Deterministic max-automata capture all ω-regular languages.*

Proof. By simulating a deterministic automaton with the Muller or parity condition. We add a new counter c_q for each state q of the automaton. Each time state q appears, counter c_q is incremented and output. The counters are never reset. In a run of this automaton, a state appears infinitely often if and only if its counter is unbounded. Therefore, the Muller acceptance condition can be encoded in the unbounding condition of a max-automaton. ∎

The induction step for boolean operations—including negation—is no more difficult, since the automata are deterministic and the accepting condition is closed under boolean operations. We are left with weak second-order quantification and the unbounding quantifier. We first deal with weak quantification, in Section 4.1, while the unbounded quantifier is treated in Section 4.2.

4.1. Weak existential quantification

This section is devoted to showing:

Proposition 4.3. *Languages recognized by deterministic max-automata are closed under weak quantification. In other words, if L is a language over $\Sigma \times \{0,1\}$ recognized by a deterministic max-automaton, then there is a deterministic max-automaton recognizing*

$$\{w \in \Sigma^\omega : w[X] \in L \text{ for some finite set } X\} .$$

A convenient way to prove this result would be to use nondeterministic automata. Unfortunately, as we will later show, adding nondeterminism to max-automata gives power beyond that of weak unbounding logic, so we cannot use this strategy. We will have to do the existential quantification directly in the deterministic automata.

The proof technique is actually very generic. It would work for any model of deterministic automata that all ω-regular languages and satisfies some relaxed assumptions, mainly that the acceptance condition is prefix-independent.

Fix a deterministic max-automaton $\mathcal{A}$ that recognizes L, with state space Q.

A *partial run* in an infinite word w is a run that begins in any position of the word (not necessarily the first position) and in any state (not necessarily the initial one). In other words, this is a word in $\bot^*\delta^\omega \cup \bot^\omega$, where δ is the set of transitions of $\mathcal{A}$, that is consistent with the word w on those positions where it is defined (i.e. where it is not $\bot$). Since the automaton is deterministic, a partial run is uniquely specified by giving the first configuration where it is defined, this is called the *seed configuration*. (There is also the undefined partial run $\bot^\omega$, which has no seed configuration.) Here, a configuration is a pair (q, x), where q is a state and x is a word position. Note that we do not include the counter

values in the seed configuration, since the acceptance condition is not sensitive to finite perturbations.

We say that two partial runs *converge* if they agree from some position on. Equivalently, they converge if they share some configuration, or both are undefined. We say a set of partial runs *spans* a word w if every partial run over w converges with some run from the set. Usually, we will be interested in finite sets of spanning runs.

Lemma 4.4. *For every word w, there is a set of at most $|Q|$ spanning runs.*

Proof. We begin with some arbitrary configuration, and take the partial run ρ_1 that begins in that configuration. If $\{\rho_1\}$ is spanning, then we are done. Otherwise, we take some partial run ρ_2 that does not converge with ρ_1, and see if the set $\{\rho_1, \rho_2\}$ is spanning. If it is not, we add a third partial run ρ_3, and so on. This process terminates after at most Q steps, because if two partial runs do not converge, then they must use different states on each position where they are both defined. So $|Q|$ partial runs that do not converge will use up all the states. ∎

To prove Proposition 4.3, we use a result stronger than Lemma 4.4. We will show that not only the spanning set of runs exists, but it can also be computed by a (deterministic, letter-to-letter) transducer. By *transducer* we mean a finite deterministic automaton where each transition is equipped with an output letter, from an output alphabet Γ. Therefore, the transducer defines a function $f : \Sigma^\omega \to \Gamma^\omega$. The transducer does not have any accepting conditions (using bounds or even parity or Muller), it just scans the word and produces its output. It is easy to see that deterministic max-automata are closed under preimages of transducers, as shown in the following lemma.

Lemma 4.5. *If f is a transducer and $\mathcal{A}$ is a deterministic max-automaton, then there is a deterministic max-automaton recognizing the set of words w such that $f(w)$ is accepted by $\mathcal{A}$.*

We now describe how the spanning partial runs will be encoded in the output of the transducer. When speaking of spanning partial runs, we mean spanning partial runs of the automaton $\mathcal{A}$ in Proposition 4.3. A single partial run can be encoded as an infinite word over the alphabet $Q \times \{0, 1\}$. The idea is that $\{0, 1\}$ is used as a marker, with 0 meaning "ignore the prefix until this position", and 1 meaning "do not ignore". Formally, an infinite word

$$(q_1, a_1)(q_2, a_2), \ldots \quad \in \quad (Q \times \{0, 1\})^\omega$$

is interpreted as the partial run which on position i has $\perp$ if $a_j = 0$ for some $j \geq i$, otherwise it has q_i. Note that if the word above has infinitely many positions j with $a_j = 0$, then the partial run is nowhere defined, i.e. it is $\perp^\infty$. If we want to encode n partial runs, we use n parallel word sequences, encoded as a single sequence over the product alphabet

$$(Q \times \{0, 1\})^n \ .$$

With the encoding of spanning runs defined, we are now ready to present the stronger version of Lemma 4.4.

Lemma 4.6. *Let $n = |Q|$. There is a transducer*

$$f : \quad \Sigma^\omega \quad \to \quad ((Q \times \{0, 1\})^n)^\omega$$

such that for any word w, the output $f(w)$ encodes n spanning partial runs.

Proof. The idea is to implement the proof of Lemma 4.4 in a transducer. The states of the transducer will be permutations of the state space, i.e. tuples from Q^n where each state appears exactly once. The initial state is any arbitrarily chosen permutation. When reading an input letter a in state $\pi = (q_1, \ldots, q_n)$, the transducer does the following operations. First, it transforms each state in π according to the letter a, giving a tuple $x = (q_1 a, \ldots, q_n a)$. This tuple is not necessarily a permutation, i.e. there are may be some coordinates $i \in \{1, \ldots, n\}$ such that the state $q_i a$ appears already in $\{q_1 a, \ldots, q_{i-1} a\}$. Let $I = \{i_1, \ldots, i_k\}$ be these coordinates, and let $\{p_1, \ldots, p_m\}$ be the states that do not appear in the new tuple x. These two sets have the same size, i.e. $k = m$. We can now correct x to be a permutation σ, by replacing its coordinate i_1 with the state p_1, the coordinate i_2 with state p_2, and so on. Note that on a the coordinates from I, the new permutation σ has a value unrelated to the one from π (i.e. σ begins a new run), while on coordinates from outside I, the new permutation σ simply continues the runs from π. This is signified in the output of the transducer, which is decorates each coordinate i of the permutation σ with a bit, which is 0 when $i \in I$ and 1 otherwise. ∎

We are now ready to prove Proposition 4.3. By properties of spanning sets of runs, a word $w \in \Sigma^\omega$ belongs to the language of the proposition if and only if there is some $i = 1, \ldots, n$ such that the following two properties hold:

(A) The i-th run encoded by $f(w)$ is defined (i.e. the encoding does not contain infinitely many cancelling 0s) and satisfies the accepting condition in the automaton $\mathcal{A}$.
(B) There is some finite set $X \subseteq \mathbb{N}$ such that the run of $\mathcal{A}$ over $w[X]$ converges with the i-th run encoded by $f(w)$.

Since deterministic max-automata are closed under union, it suffices to show that for each fixed i, both properties (A) and (B) are recognized by deterministic max-automata. For property (A), we use Lemma 4.5 on preimages. Property (B), on the other hand, is an ω-regular property, which can be recognized by a deterministic max-automaton thanks to Lemma 4.2.

4.2. Unbounding quantification

We now turn to the more difficult part of Proposition 4.1, namely that deterministic max-automata are closed under unbounding quantification.

Proposition 4.7. *Languages recognized by deterministic max-automata are closed under unbounding quantification. In other words, if L is a language over $\Sigma \times \{0,1\}$ recognized by a deterministic max-automaton, then so is*

$$UL = \{w \in \Sigma^\omega : w[X] \in L \text{ for arbitrarily large finite sets } X\} .$$

Fix a deterministic max-automaton $\mathcal{A}$ recognizing the language L in the proposition. Given a finite prefix $w \in \Sigma^*$ and a state q of $\mathcal{A}$, let $max(q, w)$ be the maximal size of a set X of positions in w such that the automaton $\mathcal{A}$ reaches state q after reading $w[X]$. We claim that the sets $max(q, w)$ can be computed in the counters of a deterministic max-automaton (not surprisingly, using the max operation).

Lemma 4.8. *There is a deterministic max-automaton with counters $\{c_q\}_{q \in Q}$ such that the value of c_q after reading a prefix $a_1 \cdots a_n$ of the input is exactly $max(q, a_1 \cdots a_n)$.*

We will use the values from the above lemma to capture the unbounding quantifier. However, some more effort is needed: it is not the case that an input word $w = a_1a_2\cdots$ belongs to UL if and only if the values $max(q, a_1\cdots a_n)$ are unbounded. In general, only the left to right implication holds. The right to left implication may fail since a value $max(q, a_1\cdots a_n)$ is relevant only if the run of $\mathcal{A}$ over w that begins in configuration (q, n) can be extended to an accepting one over the rest of the word. The correct characterization is given below:

Lemma 4.9. *A word $a_1a_2\cdots \in \Sigma^\omega$ belongs to UL if and only if for some state q, the following values are unbounded*

$\{max(q, a_1\cdots a_n) : a_{n+1}a_{n+2}\cdots[\emptyset] \in (\Sigma \times \{0,1\})^\omega$ *is accepted by $\mathcal{A}$ when starting in $q\}$*

As suggested by the above lemma, to recognize the language UL it would be convenient to have an extension of max-automata, where the automaton would have the ability to output $max(q, a_1\cdots a_n)$ only in case a certain property was satisfied by the suffix $a_{n+1}a_{n+2}\cdots$. Below, we introduce such an extension of max-automata, which we call a guarded max-automaton. We then show that this extension can be simulated by a standard max-automaton, thus completing the proof of Proposition 4.7.

An *guarded max-automaton* is like a max-automaton, except that it is also allowed to use the following counter operation:

$if\ L\ then\ output(c)$. Output the value of counter c, but only if the suffix of the input beginning at the next position belongs to $L \subseteq \Sigma^\omega$.

In the above operation, the language L—called the *guard* of the transition—must be a language recognized by a max-automaton (without guards, although allowing guards would give the same result). This new operation is all we need to recognize the language UL:

Lemma 4.10. *If a language L is recognized by a deterministic max-automaton, then UL is recognized by a deterministic guarded max-automaton.*

We will show that guarded outputs are redundant, and can be simulated by non-guarded outputs. This completes the proof Proposition 4.7. The difficulty in the proof below is that we are dealing with deterministic automata, while a guard looks to the future.

Proposition 4.11. *For every deterministic guarded max-automaton there is an equivalent deterministic max-automaton.*

Proof. Let $\mathcal{A}$ be a deterministic guarded max-automaton. To simplify notation, we assume that only one guarded operation,

$$o \quad = \quad if\ L\ then\ output(c)\ ,$$

is used. The general case is done the same way. Let $\mathcal{B}$ be a deterministic max-automaton recognizing the guard language L.

In the construction, we will use a concept of *thread*. A thread consists of a state of the automaton $\mathcal{B}$, as well as a number, which corresponds to the value of counter c output by the guarded operation o. Note that a thread does not contain information about values of the counters of automaton $\mathcal{B}$. The idea is that threads will be alive for only finitely many steps, so the counters of $\mathcal{B}$ are not relevant. We will denote threads by τ. If $a \in \Sigma$ is an input letter, then we write τa for the thread obtained from τ by updating the state according to a (and leaving the number unchanged).

The (non-guarded) max-automaton $\mathcal{C}$ that simulates $\mathcal{A}$ works as follows. At each point, the simulating automaton contains a finite set $\{\tau_1, \ldots, \tau_i\}$ of *active* threads. There will be at most one thread per state of $\mathcal{B}$, so the set of threads can be stored using finitely many counters and the finite memory of the automaton. This set of active threads is initially empty. Whenever $\mathcal{A}$ does the guarded output operation o, a new active thread is created, with the initial state of $\mathcal{B}$, and the number set to the value of counter c. Furthermore, after reading an input letter $a \in \Sigma$, the set of active threads is updated to $\{\tau_1 a, \ldots, \tau_i a\}$. If two active threads have the same state, then they are merged, and only the greater number is kept (using the max operation).

Similarly to the proof of Proposition 4.3, the automaton $\mathcal{C}$ will also read the output of a transducer f that computes spanning partial runs of the automaton $\mathcal{B}$ used for the guards. Recall that the transducer f outputs n spanning partial runs of the automaton $\mathcal{B}$, where n is the number of states in $\mathcal{B}$.

The automaton $\mathcal{C}$ accepts a word w if and only if there is some $i = 1, \ldots, n$ such that:

(A) The i-th run encoded by $f(w)$ is defined (i.e. the encoding does not contain infinitely many cancelling 0s) and satisfies the accepting condition in the automaton $\mathcal{B}$.
(B) For every m, some thread storing a number greater than m converges with i-th run encoded by $f(w)$.

Since deterministic max-automata are closed under finite union, we only need to show the construction for some fixed i. As in the previous section, property (A) is recognized by a deterministic max-automaton. For property (B), it suffices to output the number stored in a thread τ whenever its state is the same as in ρ_i. The automaton then accepts if the numbers thus produced are unbounded.

■

5. Problems with nondeterminism

In this section we show that nondeterministic max-automata are more expressive than deterministic ones.

Theorem 5.1. *Nondeterministic max-automata recognize strictly more languages than deterministic ones.*

Contrast this result with the situation for Muller or parity automata, which are equally expressive in the deterministic and nondeterministic variants. Since full monadic second-order can capture nondeterministic automata by existentially quantifying over infinite sets, the above theorem immediately implies:

Corollary 5.2. *Full monadic second-order logic with the unbounding quantifier is stronger than weak monadic second-order with the unbounding quantifier.*

The separating language in Theorem 5.1 is

$$L = \{a^{n_1} b a^{n_2} b a^{n_3} b \ldots : \text{ some number appears infinitely often in } n_1, n_2, \ldots\} . \tag{5.1}$$

This language is captured by a nondeterministic max-automaton. The automaton uses nondeterminism to output a subsequence of $n_1, n_2, \ldots$ and accepts if this subsequence is bounded. Clearly, if it is bounded, then it contains an infinite constant subsequence.

It remains to show that the language L cannot be recognized by a deterministic max-automaton. For this, we will use topological complexity. In Lemmas 5.3 and 5.4, we

will show that every language recognized by a deterministic max-automaton is a boolean combination of sets on level Σ_2 in the Borel hierarchy, while the language L is not.

Below we briefly describe the Borel hierarchy, a way of measuring the complexity of a subset of a topological space. The topology that we use on words is that of the Cantor space, as described below. A set of infinite words (over a given alphabet Σ) is called *open* if it is a union

$$\bigcup_{i \in I} w_i \Sigma^\omega \qquad w_i \in \Sigma^* \ ,$$

with the index set I being possibly infinite. In other words, membership of a word w in an open set is assured already by a finite prefix of w. For the Borel hierarchy, as far as max-automata are concerned, we will only be interested in the first two levels $\Sigma_1, \Pi_1, \Sigma_2, \Pi_2$. The open subsets are called Σ_1, the complements of these (the closed subsets) are called Π_1. Countable intersections of open subsets are called Π_2, the complements of these (countable unions of closed subsets) are called Σ_2.

Lemma 5.3. *Any language accepted by a deterministic max-automaton is a boolean combination of Σ_2 sets.*

Proof. Fix a max-automaton $\mathcal{A}$, and a counter c of this automaton. We will examine the topological complexity of the set of runs of this automaton (here, a run is an infinite sequence of transitions). For any fixed n, the following set of runs is clearly open:

A value of at least n is output at least once on counter c.

In particular, its complement

All values of counter c are at most n.

is a closed set of runs. By taking a countable union of the above over $n \in \mathbb{N}$, we deduce that the property

The values of counter c are bounded.

is a Σ_2 property. In particular, the set of accepting runs of any max-automaton is a boolean combination of Σ_2 sets. Since the automata are deterministic, the function that maps an input word to its run is continuous, i.e. preimages of open sets are also open. Since preimages of continuous functions preserve the levels of the hierarchy, we conclude that any language accepted by a deterministic max-automaton is a boolean combination of Σ_2 sets. ∎

Lemma 5.4. *The language L is not a boolean combination of Σ_2 sets.*

Proof. Consider the mapping from $\mathbb{N}^*$ to $\{a, b\}^*\omega$ defined by

$$n_1, n_2, \ldots\ldots \qquad \mapsto \qquad a^{n_1} b a^{n_2} b a^{n_3} b \ldots$$

This is a continuous mapping. The language L is the image, under this mapping, of the set X of sequences in $\mathbb{N}^\omega$ that have a bounded subsequence. The set X is known not to be a boolean combination of Σ_2 sets, see Excercise 23.2 in [9]. ∎

6. Conclusion

This paper is intended as a proof of concept. The concept is that ω-regular languages can be extended in various ways, while still preserving good closure properties and decidability. The class presented in this paper, max-regular languages, is closed under boolean operations, inverse morphisms, and quotients. It is not closed under morphic images (which corresponds to nondeterminism on the automaton side).

Some questions on max-automata are left unresolved. Is the max operation necessary in the automaton? In our construction, we use the max twice: when defining the values $max(q, a_1 \cdots a_n)$, and in Proposition 4.11. While in the first case, the max operation can be avoided by a subtle use of factorization forests [12], it is not clear how to show Proposition 4.11 without using the max operation. Another question is the exact complexity of emptiness. It would be nice to get matching upper and lower bounds, even more so if the lower bound would use acceptance conditions in DNF.

There are several other possibilities of future work. One is to investigate weak bounding logic for infinite trees (note that we will not capture all regular languages of infinite trees in this case). Another possibility would be to investigate full monadic-second order logic, or possibly other quantifiers that can be added to weak monadic second-order logics. The techniques used in this paper are fairly generic, so it seems plausible that such quantifiers can be found.

References

[1] P. A. Abdulla, P. Krcál, and W. Yi. R-automata. In *CONCUR*, pages 67–81, 2008.
[2] A. Arnold. A syntactic congruence for rational omega-language. *Theor. Comput. Sci.*, 39:333–335, 1985.
[3] M. Bojańczyk. A bounding quantifier. In *Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 41–55, 2004.
[4] M. Bojańczyk and T. Colcombet. Omega-regular expressions with bounds. In *Logic in Computer Science*, pages 285–296, 2006.
[5] T. Colcombet and C. Löding. The nesting-depth of disjunctive mu-calculus for tree languages and the limitedness problem. In *Computer Science Logic*, volume 5213 of *Lecture Notes in Computer Science*, 2008.
[6] T. Colcombet and C. Löding. The non-deterministic mostowski hierarchy and distance-parity automata. In *International Colloquium on Automata, Languages and Programming*, volume 5126 of *Lecture Notes in Computer Science*, pages 398–409, 2008.
[7] M. Droste and P. Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007.
[8] K. Hashiguchi. Algorithms for determining relative star height and star height. *Inf. Comput.*, 78(2):124–169, 1988.
[9] A. S. Kechris. *Classical Descriptive Set Theory*, volume 156 of *Graduate Texts in Mathematics*. Springer, 1995.
[10] D. Kirsten. Distance desert automata and the star height problem. *Theoretical Informatics and Applications*, 39(3):455–511, 2005.
[11] M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
[12] I. Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72:65–94, 1990.
[13] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 171–182
www.stacs-conf.org

POLYNOMIAL-TIME APPROXIMATION SCHEMES FOR SUBSET-CONNECTIVITY PROBLEMS IN BOUNDED-GENUS GRAPHS

GLENCORA BORRADAILE [1] AND ERIK D. DEMAINE [2] AND SIAMAK TAZARI [3]

[1] University of Waterloo
E-mail address: glencora@uwaterloo.ca

[2] Massachusetts Institute of Technology
E-mail address: edemaine@mit.edu

[3] Technische Universität Darmstadt
E-mail address: tazari@cs.tu-darmstadt.de

Abstract. We present the first polynomial-time approximation schemes (PTASes) for the following subset-connectivity problems in edge-weighted graphs of bounded genus: Steiner tree, low-connectivity survivable-network design, and subset TSP. The schemes run in $O(n \log n)$ time for graphs embedded on both orientable and non-orientable surfaces. This work generalizes the PTAS frameworks of Borradaile, Klein, and Mathieu [BMK07, Kle06] from planar graphs to bounded-genus graphs: any future problems shown to admit the required structure theorem for planar graphs will similarly extend to bounded-genus graphs.

1. Introduction

In many practical scenarios of network design, input graphs have a natural drawing on the sphere or equivalently the plane. In most cases, these embeddings have few crossings, either to avoid digging multiple levels of tunnels for fiber or cable or to avoid building overpasses in road networks. But a few crossings are common, and can easily come in bunches where one tunnel or overpass might carry several links or roads. Thus we naturally arrive at graphs of small (bounded) genus, which is the topic of this paper.

We develop a PTAS framework for subset-connectivity problems on edge-weighted graphs of bounded genus. In general, we are given a subset of the nodes, called *terminals*, and the goal is to connect the terminals together with some substructure of the graph by using cost within $1+\varepsilon$ of the minimum possible cost. Our framework applies to three well-studied problems in this framework. In *Steiner tree*, the substructure must be connected, and thus forms a tree. In *subset TSP (Traveling Salesman Problem)*, the substructure must be a cycle; to guarantee existence, the cycle may traverse vertices and edges multiple

Key words and phrases: polynomial-time approximation scheme, bounded-genus graph, embedded graph, Steiner tree, survivable-network design, subset TSP.

Glencora Borradaile was supported by an NSERC Postdoctoral Fellowship; Siamak Tazari was supported by the Deutsche Forschungsgemeinschaft (DFG), grant MU1482/3-1.

times, but pays for each traversal. In $\{0,1,2\}$-*edge-connectivity survivable network design*, the substructure must have $\min\{c_x, c_y\}$ edge-disjoint paths connecting terminals x and y, where each $c_x \in \{0,1,2\}$; we allow the substructure to include multiple copies of an edge in the graph, but pay for each copy. In particular, if $c_x = 1$ for all terminals x and y, then we obtain the Steiner tree problem; if $c_x = 2$ for all terminals x and y, then we obtain the *minimum-cost* 2-*edge-connected submultigraph* problem.

Our framework yields the first PTAS for all of these problems in bounded-genus graphs. These PTASs are efficient, running in $O(f(\varepsilon, g)\, n + h(g)\, n \log n)$ time for graphs embedded on orientable surfaces and non-orientable surfaces. (We usually omit the mention of $f(\varepsilon, g)$ and $h(g)$ by assuming ε and g are constant, but we later bound $f(\varepsilon, g)$ as singly exponential in a polynomial in $1/\varepsilon$ and g and $h(g)$ as singly exponential in g.) In contrast, the problems we consider are APX-complete (and $O(1)$-approximable) for general graphs.

We build upon recent PTAS framework of Borradaile, Klein, and Mathieu [BMK07] for subset-connectivity problems on planar graphs. In fact, our result is strictly more general: any problem to which the previous planar-graph framework applies automatically works in our framework as well, resulting in a PTAS for bounded-genus graphs. For example, Borradaile, Klein and Pritchard [BKP] have recently claimed a PTAS for the $\{0,1,\ldots,k\}$-*edge-connectivity survivable network design* problem using the planar framework. This will imply a similar result in bounded genus graphs. In contrast to the planar-graph framework, our PTASes have the attractive feature that they run correctly on all graphs with the performance degrading with genus.

Our techniques for attacking bounded-genus graphs include two recent results from SODA 2007: decompositions into bounded-treewidth graphs via contractions [DHM07] and fast algorithms for finding the shortest noncontractible cycle [CC07]. We also use a simplified version of an algorithm for finding a short sequence of loops on a topological surface from SODA 2005 [EW05], and sophisticated dynamic programming.

2. Basics

All graphs $G = (V, E)$ have n vertices, m edges and are undirected with edge lengths (weights). The length of an edge e, subgraph H, and set of subgraphs $\mathcal{H}$ are denoted $\ell(e)$, $\ell(H)$ and $\ell(\mathcal{H})$, respectively. The shortest distance between vertices x and y in graph G is denoted $\mathrm{dist}_G(x, y)$. The boundary of a graph G embedded in the plane is denoted by ∂G. For an edge $e = uv$, we define the operation of *contracting* e as identifying u and v and removing all loops and duplicate edges.

We use the basic terminology for embeddings as outlined in [MT01]. In this paper, an embedding refers to a *2-cell embedding*, i.e. a drawing of the vertices and faces of the graph as points and arcs on a surface such that every face is homeomorphic to an open disc. Such an embedding can be described purely combinatorially by specifying a *rotation system*, for the cyclic ordering of edges around vertices of the graph, and a *signature* for each edge of the graph; we use this notion of a *combinatorial embedding*. A combinatorial embedding of a graph G naturally induces such a 2-cell embedding on each subgraph of G. We only consider compact surfaces without boundary. When we refer to a planar embedding, we actually mean an embedding in the 2-sphere. If a surface contains a subset homeomorphic to a Möbius strip, it is *non-orientable*; otherwise it is *orientable*. For a 2-cell embedded graph G with f facial walks, the number $g = 2 + m - n - f$ is called the Euler genus of the surface. The Euler genus is equal to twice the usual genus for orientable surfaces and

equals the usual genus for non-orientable surfaces. The *dual* of an embedded graph G is defined as having the set of faces of G as its vertex set and having an edge between two vertices if the corresponding faces of G are adjacent. We denote the dual graph by $G^\star$ and identify each edge of G with its corresponding edge in $G^\star$. A cycle of an embedded graph is *contractible* if it can be continuously deformed to a point; otherwise it is *non-contractible*. The operation of *cutting along a 2-sided cycle* C is essentially: partition the edges adjacent to C into left and right edges and replace C with two copies C_ℓ and C_r, adjacent to the left or right edges, accordingly. The inside of these new cycles is "patched" with two new faces. If the resulting graph is disconnected, the cycle is called *separating*, otherwise *non-separating*. Cutting along a 1-sided cycle C on non-orientable surfaces is defined similarly, only that C is replaced by one bigger cycle C' that contains every edge of C exactly twice.

Next we define the notions related to treewidth as introduced by Robertson and Seymour [RS86]. A *tree decomposition* of a graph G is a pair (T, χ), where $T = (I, F)$ is a tree and $\chi = \{\chi_i | i \in I\}$ is family of subsets of $V(G)$, called *bags*, such that

(1) every vertex of G appears in some bag of χ;
(2) for every edge $e = uv$ of G, there exists a bag that contains both u and v;
(3) for every vertex v of G, the set of bags that contain v form a connected subtree T_v of T.

The *width* of a tree decomposition is the maximum size of a bag in χ minus 1. The *treewidth* of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum width over all possible tree decompositions of G.

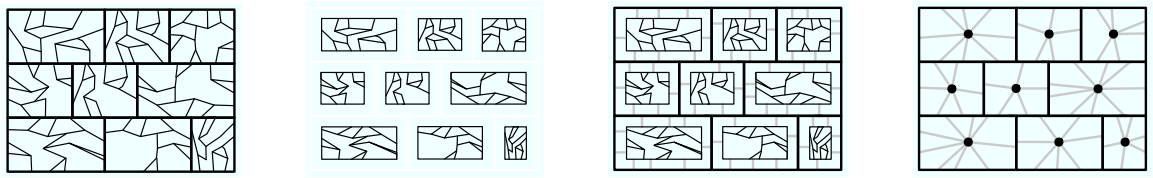
The input graph is $G_0 = (V_0, E_0)$ and has genus g_0; the terminal set is Q. We assume G_0 is equipped with a combinatorial embedding; such an embedding can be found in linear time, if the genus is known to be fixed, see [Moh99]. Let $\mathcal{P}$ be the considered subset-connectivity problem. In Section 5.1, we show how to find a subgraph $G = (V, E)$ of G_0, so that for $0 \le \varepsilon \le 1$ any $(1+\varepsilon)$-approximate solution of $\mathcal{P}$ in G_0 also exists in G. Hence, we may use G instead of G_0 in the rest of the paper. Note that as a subgraph of G_0, G is automatically equipped with a combinatorial embedding.

Let OPT denote the length of an optimal Steiner tree spanning terminals Q. We define $\mathrm{OPT}_\mathcal{P}$ to be the length of an optimal solution to problem $\mathcal{P}$. For the problems that we solve, we require that $\mathrm{OPT}_\mathcal{P} = \Theta(\mathrm{OPT})$ and in particular that $\mathrm{OPT} \le \mathrm{OPT}_\mathcal{P} \le \xi\,\mathrm{OPT}$. The constant ξ will be used in Section 5 and is equal to 2 for both the subset TSP and $\{0,1,2\}$-edge-connectivity problems. This requirement is also needed for the planar case; see [BK08]. Because $\mathrm{OPT}_\mathcal{P} \ge \mathrm{OPT}$, upper bounds in terms of OPT hold for all the problems herein. As a result, we can safely drop the $\mathcal{P}$ subscript throughout the paper.

We show how to obtain a $(1 + c\varepsilon)\,\mathrm{OPT}_\mathcal{P}$ solution for a fixed constant c. To obtain a $(1+\varepsilon)\,\mathrm{OPT}_\mathcal{P}$ solution, we can simply use $\varepsilon' = \varepsilon/c$ as input to the algorithm.

3. Structure Theorem

In [BMK07] and [BMK07b], Borradaile, Klein and Mathieu developed a PTAS for the Steiner tree problem in planar graphs. The method involves finding a grid-like subgraph called the *mortar graph* that spans the input terminals and has length $O(\mathrm{OPT})$. The set of feasible Steiner trees is restricted to those that cross between adjacent faces of the mortar graph only at a small number (per face of the mortar graph) of pre-designated vertices called *portals*. A Structure Theorem guarantees the existence of a nearly optimal solution (one



(a) (b) (c) (d)

Figure 1: (a) An input graph G with mortar graph MG given by bold edges. (c) The set of bricks corresponding to MG (d) A portal-connected graph, $\mathcal{B}^+(MG, \theta)$. The portal edges are grey. (e) $\mathcal{B}^+(MG, \theta)$ with the bricks contracted, resulting in $\mathcal{B}^\div(MG, \theta)$. The dark vertices are brick vertices.

that has length at most $(1+\varepsilon)\,\mathrm{OPT}$) in this set. We review the details that are relevant to this work and generalize to genus-g graphs.

3.1. Mortar Graph

Here we define the mortar graph in such a way that generalizes to higher genus graphs. A path P in a graph G is *t-short in G* if for every pair of vertices x and y on P, the distance from x to y along P is at most $(1+t)$ times the distance from x to y in G: $\mathrm{dist}_P(x,y) \le (1+t)\mathrm{dist}_G(x,y)$. Given a graph G embedded on a surface and a set of terminals Q, a mortar graph is a subgraph of G with the following properties (where κ, to be defined later, will depend polynomially on ε^{-1} and g):

Definition 3.1 (Mortar Graph and Bricks)**.** Given an embedded graph G and a set of terminals Q, a mortar graph is a subgraph MG of G spanning Q such that each facial walk of MG encloses an area homeomorphic to an open disk. For each face of MG, we construct a *brick* B of G by cutting G along the facial walk. B is the subgraph of G embedded inside the face, including the facial walk. Each brick satisfies the following properties:

(1) B is planar.
(2) The boundary of B is the union of four paths in the clockwise order W, N, E, S.
(3) Every terminal of Q that is in B is on N or on S.
(4) N is 0-short in B, and every proper subpath of S is ε-short in B.
(5) There exists a number $k \le \kappa$ and vertices $s_0, s_1, s_2, \ldots, s_k$ ordered from left to right along S such that, for any vertex x of $S[s_i, s_{i+1})$, the distance from x to s_i along S is less than ε times the distance from x to N in B: $\mathrm{dist}_S(x, s_i) < \varepsilon\,\mathrm{dist}_B(x, N)$.

The mortar graph and the set of bricks are illustrated in Figures 1 (a), (b) and (c). Constructing the mortar graph for planar graphs first involves finding a 2-approximate Steiner tree T and cutting open the graph along T creating a new face with boundary H and then:

(1) Finding shortest paths between certain vertices of H. These paths result in the N and S boundaries of the bricks.
(2) Finding shortest paths between vertices of the paths found in Step 1. These paths are called *columns*, do not cross each other, and have a natural order.
(3) Taking every κth path found in Step 2. These paths are called *supercolumns* and form the E and W boundaries of the bricks.

The mortar graph is composed of the edges of T (equivalently, H) and the edges found in Steps 1 and 3. In [BMK07], it is shown that the total length of the mortar graph edges is $9\varepsilon^{-1}\,\mathrm{OPT}$[1]. For the purposes of this paper, we bound the length of the mortar graph in terms of $\ell(H)$. The following theorem can be easily deduced from [Kle06] and [BMK07]:

Theorem 3.2. [Kle06, BMK07] *Given a planar graph with boundary H containing the terminals and a number κ, there is a mortar graph containing H whose length is at most $(\varepsilon^{-1}+1)^2\ell(H)$ and whose supercolumns have length at most $\frac{1}{\kappa}\varepsilon^{-1}(\varepsilon^{-1}+1)\ell(H)$. The mortar graph can be found in $O(n\log n)$ time.*

We will use this theorem to prove the existence of a mortar graph for genus-g embedded graphs. Section 5 is devoted to proving the following theorem:

Theorem 3.3. *Let G be an embedded graph with fixed genus g. Let Q be a subset of vertices. For $\gamma = 2(8\xi g+2)(\varepsilon^{-1}+1)^2$ and $\kappa = \gamma/\varepsilon$, there is a mortar graph MG of G such that the length of MG is $\le \gamma\,\mathrm{OPT}$ and the supercolumns of MG have length $\le \varepsilon\,\mathrm{OPT}$. The mortar graph can be found in $O(n\log n)$ time.*

3.2. Structural properties

Along with the mortar graph, Borradaile et al. [BMK07b] define an operation $\mathcal{B}^+$ called *brick-copy* that allows a succinct statement of the Structure Theorem. For each brick B, a subset of θ vertices are selected as *portals* such that the distance along ∂B between any vertex and the closest portal is at most $\ell(\partial B)/\theta$. For every brick B, embed B in the corresponding face of MG and connect every portal of B to the corresponding vertex of MG with a zero-length *portal edge*: this defines $\mathcal{B}^+(MG,\theta)$. $\mathcal{B}^+(MG,\theta)$ is illustrated in Figure 1 (d). We denote the set of all portal edges by {portal edges}. The following simple lemma, proved in [BMK07b] holds also for bounded-genus graphs:

Lemma 3.4. [BMK07b] *If A is a connected subgraph of $\mathcal{B}^+(MG,\theta)$, then $A\setminus\{\text{portal edges}\}$ is a connected subgraph of G spanning the same nodes of G.*

The following Structure Theorem is the heart of the correctness of the PTASs.

Theorem 3.5 (Structure Theorem)**.** *Let G be an embedded graph of genus g. There exists a constant $\theta(\gamma,\alpha)$ depending polynomially on γ and α such that*

$$\mathrm{OPT}(\mathcal{B}^+(MG,\theta),Q) \le (1+c\varepsilon)\,\mathrm{OPT}(G,Q).$$

Here $\alpha = o(\varepsilon^{-2.5}\kappa)$ for Steiner tree and $\{0,1,2\}$-edge connectivity and $\alpha = O(\kappa)$ for subset TSP. Recall that κ and γ depend polynomially on ε^{-1} and g (Theorem 3.3).

Since the bricks are always planar, the proof for bounded-genus graphs follows as for the planar cases: see [BMK07b] for the Steiner tree problem and [BK08] for the $\{0,1,2\}$-edge-connectivity problem. In the remainder of this section, we sketch the proof for the subset-TSP problem.

We will use the following lemma due to Arora:

Lemma 3.6 (Patching Lemma [Aro03])**.** *Let S be any line segment of length s and π be a closed path that crosses S at least thrice. Then we can break the path in all but two of these places and add to it line segments lying on S of total length at most $3s$ such that π changes into a closed path π' that crosses S at most twice.*

[1]Actually, they claim that the length is $5\varepsilon^{-1}\,\mathrm{OPT}$. The correction is forthcoming in a journal version.

This lemma applies to embedded graphs as well. Note: the patching process connects paths in the tour that end on a common side of S by a subpath of S.

Let H be a subgraph of a graph G and let P be a path in H. A *joining vertex* of H with P is a vertex of P that is the endpoint of an edge of $H \setminus P$. The proof of Theorem 3.2 for subset-TSP follows from the following TSP Property of Bricks using the same technique as the planar case for, e.g., Steiner tree. See [BMK07b] for details. The condition that E and W are crossed at most twice each is achieved by including two copies of each supercolumn in the solution and rerouting the tour around these copies.

Theorem 3.7 (TSP Property of Bricks)**.** *Let B be a brick of graph G with boundary $N \cup E \cup S \cup W$ (where E and W are supercolumns). Let T be a tour in G such that T crosses E and W at most 2 times each. Let H be the intersection of T with B. Then there is another subgraph of B, H', such that:*

(T1) H' has at most $\alpha(\varepsilon)$ joining vertices with ∂B.
(T2) $\ell(H') \leq (1+3\varepsilon)\ell(H)$.
(T3) There is a tour in the edge set $T \setminus H \cup H'$ that spans the vertices in $\partial B \cap T$.
In the above, $\alpha(\varepsilon) = O(\kappa)$.

Proof. Let H be the subgraph of T that is strictly enclosed by B (i.e., H contains no edges of ∂B). We can write H as the union of 4 sets of minimal ∂B-to-∂B paths $\mathcal{P}_S \cup \mathcal{P}_N \cup \mathcal{P}_{EW} \cup \mathcal{P}_{NS}$ where paths in: $\mathcal{P}_S$ start and end on S; $\mathcal{P}_N$ start and end on N; $\mathcal{P}_{EW}$ start on E or W and end on ∂B; $\mathcal{P}_{NS}$ start on S and end on N.

Because T crosses E and W at most 4 times, $|\mathcal{P}_{EW}| \leq 4$ and $\mathcal{P}_{EW}$ results in at most 8 joining vertices with ∂B. For each path $P \in \mathcal{P}_N$, let $\widehat{P}'$ be the minimal subpath of N that spans P's endpoints and let $\widehat{\mathcal{P}}_N$ be the resulting set. Similarly define $\widehat{\mathcal{P}}_S$. Because N is 0-short and S is ε-short,

$$\ell(\widehat{\mathcal{P}}_N) + \ell(\widehat{\mathcal{P}}_S) \leq \ell(\mathcal{P}_N) + (1+\varepsilon)\ell(\mathcal{P}_S). \tag{3.1}$$

Because $\widehat{\mathcal{P}}_N$ and $\widehat{\mathcal{P}}_S$ are subpaths of ∂B, they result in no joining vertices with ∂B.

It remains to deal with the paths in $\mathcal{P}_{NS}$.

Let $s_0, s_1, s_2, \ldots, s_k$ (where $k \leq \kappa$) be the vertices of S guaranteed by the properties of the bricks (see Section 3.1). Let $\mathcal{X}_i$ be the subset of paths of $\mathcal{P}_{NS}$ that start on S strictly between s_i and s_{i+1}. Let $\mathcal{X}$ be the remaining paths (note: $|\mathcal{X}| \leq \kappa + 1$).

If $|\mathcal{X}_i| > 2$, we do the following: Let P_i be the path in $\mathcal{X}_i$ whose endpoint x on S is closest to s_{i+1}. Let Q_i be the subpath of S from s_i to x. By the properties of the bricks, $\ell(Q_i) \leq \varepsilon\ell(P_i)$. Apply the Patching Lemma to the tour T and path Q_i: as a result, at most two paths of $\mathcal{X}_i$ occur right before (or after) crossings of Q in the new tour T'. Let $\mathcal{X}'_i$ whose endpoints are not crossings in T': $|\mathcal{X}'_i| \leq |\mathcal{X}_i| - 2$. Let $\mathcal{Q}_i$ be the subpaths of Q_i that are added to the tour. By the Patching Lemma,

$$\ell(\mathcal{Q}_i) \leq 3\ell(Q_i) \leq 3\varepsilon\ell(P_i). \tag{3.2}$$

While $|\mathcal{X}'_i| \geq 2$, we do the following: Let P be any path in $\mathcal{X}'_i$. As a result of the patching process P is connected to another path P' in $\mathcal{X}_i$ via a path $Q' \in \mathcal{Q}_i$: that is, $P \cup Q' \cup P'$ is a subpath of the tour. Let L be the minimal subpath of N connecting the endpoints of P and P'. Because N is 0-short, $\ell(L) \leq \ell(P) + \ell(Q') + \ell(P')$. By the patching process, the endpoints of P and P' on S are spanned by a part of the tour "on the other side" of Q_i. Remove Q_i from $\mathcal{Q}_i$ and add L. Remove P and P' from $\mathcal{X}_i$. When we are done, $\mathcal{X}'_i$ is empty.

Let $\widehat{\mathcal{Q}}_i$ be the set resulting from all such replacements. Let $\widehat{\mathcal{P}}_{NS}$ be the union of $\mathcal{X}$, $\mathcal{X}_i \setminus \mathcal{X}_i'$ and $\widehat{\mathcal{Q}}_i$ over all i. Because $\widehat{\mathcal{Q}}_i$ is a set of subpaths of N and S, these paths result in no joining vertices with ∂B. Because $|\mathcal{X}_i \setminus \mathcal{X}_i'| \leq 2$, these paths result in at most 4 joining vertices with ∂B. Because $|\mathcal{X}| \leq \kappa + 1$, these paths have at most $2(\kappa + 1)$ joining vertices with ∂B. Therefore $\widehat{\mathcal{P}}_{NS}$ has at most $6(\kappa + 1)$ joining vertices with ∂B.

Because the only added length is introduced via the patching process, by Equation (3.2),

$$\ell(\widehat{\mathcal{P}}_{NS}) \leq \ell(\mathcal{P}_{NS}) + \sum_i 3\varepsilon \ell(P_i) \leq (1 + 3\varepsilon)\ell(\mathcal{P}_{NS}). \tag{3.3}$$

Let H' be the union of the paths in $\mathcal{P}_{EW}$, $\widehat{\mathcal{P}}_S$, $\widehat{\mathcal{P}}_N$, and $\widehat{\mathcal{P}}_{NS}$. Combining Equations (3.1) and (3.3), we find that $\ell(H') \leq (1 + 3\varepsilon)\ell(H)$. By construction, the edges in $T \setminus H \cup H'$ contains a tour. H' has $6(\kappa + 1) + 8 = 6\kappa + 14$ joining vertices with ∂B. ∎

4. Algorithm

In [Kle06], Klein gave a framework for designing PTASes in planar graphs that is based on finding a *spanner* for a problem, a subgraph containing a nearly optimal solution having length $O(\mathrm{OPT})$. It is possible, using the techniques in this paper and in [DHM07], to find such a spanner for bounded-genus graphs. We omit the details in favour of generalizing the framework of Borradaile et al. [BMK07b]. While both methods result in $O(n \log n)$ algorithms, the first method is doubly exponential in a polynomial in g and ε^{-1} and the second is singly exponential.[2]

The idea in [BMK07b] is to perform dynamic programming over the bricks of the brick decomposition after performing a *thinning* step which groups the bricks into manageable groups. To this end, the operation *brick-contraction* $\mathcal{B}^{\div}$ is defined to be the application of the operation $\mathcal{B}^{+}$ followed by contracting each brick to become a single vertex of degree at most θ (see Figure 1(e)). The thinning algorithm decomposes the mortar graph MG into parts, called *parcels*, of *bounded dual radius* (implying bounded treewidth). Applying $\mathcal{B}^{\div}$ to each parcel maintains bounded dual radius. The dynamic program computes optimal Steiner trees inside the bricks using the method of [EMAFV87] only at the leaves of the dynamic programming tree, thus eliminating the need of an a-priori constructed spanner. The interaction between subproblems of the dynamic programming is restricted to the portals, of which there are a few.

For embedded graphs with genus > 0, the concept of bounded dual radius does not apply in the same way. We deal with treewidth directly by applying the following Contraction Decomposition Theorem due to Demaine et al. [DHM07]:

Theorem 4.1. [DHM07, Theorem 1.1] *For a fixed genus g, and any integer $k \geq 2$ and for every graph G of Euler genus at most g, the edges of G can be partitioned into k sets such that contracting any one of the sets results in a graph of treewidth at most $O(g^2 k)$. Furthermore, such a partition can be found in $O(g^{5/2} n^{3/2} \log n)$ time.*

[2]For the subset-TSP problem, it is possible to obtain a singly exponential algorithm by following the spanner construction of Klein [Kle06] after performing the planarizing step (Lemma 5.2). Our presentation here is chosen to unify the methods for all problems studied.

(Recent techniques [CC07] have improved the above running time to $O(n \log n)$.)

We apply the above theorem directly to $\mathcal{B}^{\div}(MG)$ and contract a set of edges $S^\star$ in $\mathcal{B}^{\div}(MG)$. In $\mathcal{B}^{\div}(MG)$, we modify the definition of contraction: after contracting an edge, we do not delete parallel portal edges. Because portal edges connect the mortar graph to the bricks, they are not parallel in the graph in which we find a solution via dynamic programming. With this modified definition, we have the following algorithm:

THINNING(G, MG):

(1) Assign the weight $\ell(\partial F)$ to each portal edge e enclosed in a face F of MG.
(2) Apply the contraction decomposition of Theorem 4.1 to $\mathcal{B}^{\div}(MG)$ with $k := 3\theta\gamma\varepsilon^{-1}$ to obtain edge sets $S_1, \ldots, S_k$; let $S^\star$ be the set of minimum weight.
(3) If $S^\star$ includes a portal edge e of a brick B enclosed in a face F of MG, add ∂F to $S^\star$ and mark B as ignored.
(4) Let $K := \mathcal{B}^{\div}(MG)/S^\star$ and (T, χ) be a tree decomposition of width $O(g^2k)$ of K.
(5) For each vertex b of K that represents an unignored contracted brick with portals $\{p_1, \ldots, p_\theta\}$:
    5.1. Replace every occurence of b in χ with $\{p_1, \ldots, p_\theta\}$;
    5.2. Add a bag $\{b, p_1, \ldots, p_\theta\}$ to χ and connect it to a bag containing $\{p_1, \ldots, p_\theta\}$.
(6) Reset the weight of the portal edges back to zero.
(7) Return (T, χ) and $S^\star$.

Lemma 4.2. *The algorithm* THINNING(G, MG) *returns a set of edges $S^\star$ and a tree decomposition (T, χ) of $\mathcal{B}^{\div}(MG)$, so that*

(i) the treewidth of (T, χ) is at most ω where $\omega(\varepsilon^{-1}, g) = O(g^2k\theta)$;
(ii) every brick is either marked as ignored or none of its portal edges are in $S^\star$;
(iii) $\ell(S^\star) \le \varepsilon\,\mathrm{OPT}$;

Proof. We first verify that (T, χ) is indeed a tree decomposition. For a vertex v and a tree decomposition (T', χ'), let T'_v denote the subtree of T' that contains v in all of its bags. Let us denote the tree decomposition of step (4) by (T^0, χ^0). For each brick vertex b and each of its portals p_i, we know that T^0_b is connected and $T^0_{p_i}$ is connected and that these two subtrees intersect; it follows that after the replacement in step (5.2), we have that $T_{p_i} = T^0_b \cup T^0_{p_i}$ is a connected subtree of T and hence, (T, χ) is a correct tree decomposition. Note that Theorem 4.1 guarantees a tree decomposition of width $O(g^2k)$ if any of $S_1, \ldots, S_k$ are contracted; and in step (3), we only add to the set of edges to be contracted. Hence, the treewidth of (T^0, χ^0) is indeed $O(g^2k)$ and with the construction in line (5.1), the size of each bag will be multiplied by a factor of at most θ. This shows the correctness of claim (i). The correctness of claim (ii) is immediate from the construction in line (3). It remains to verify claim (iii).

Let L denote the weight of $\mathcal{B}^{\div}(MG)$ after setting the weights of the portal edges according to step (1) of the algorithm. We have that

$$L \le \ell(MG) + \sum_F \ell(\partial F)\theta \le \gamma\,\mathrm{OPT} + \theta \sum_F \ell(\partial F) \le \gamma\,\mathrm{OPT} + \theta \cdot 2\gamma\,\mathrm{OPT} \le 3\theta\gamma\,\mathrm{OPT}\,.$$

Hence, the weight of $S^\star$, as selected in step (2), is at most $L/k \le \frac{3\theta\gamma\,\mathrm{OPT}}{3\theta\gamma\varepsilon^{-1}} \le \varepsilon\,\mathrm{OPT}$. The operation in step (3) does not add to the weight of $S^\star$: when the boundary of a face F is

added to $S^\star$, its weight is subtracted again when resetting the weights of the portal edges back to 0 in step (6). ∎

If a brick is "ignored" by THINNING, the boundary of its enclosing mortar graph face is completely added to $S^\star$. Because $S^\star$ can be added to the final solution, every potential connection through that brick can be rerouted through $S^\star$ around the boundary of the brick. The interior of the brick is not needed.

An almost standard dynamic programming algorithm for bounded-treewidth graphs (cf. [AP89, KS90]) can be applied to K and (T, χ). However, for the leaves of the tree decomposition that are added in step (5.2) of the THINNING procedure, the cost of a subset of portal edges is calculated as, e.g., the cost of the minimum Steiner tree interconnecting these portals in the corresponding brick. Because the bricks are planar, this cost can be calculated by the algorithm of Erickson et al. [EMAFV87] for Steiner tree or [BK08] for 2-edge connectivity. Because all the portal edges of this brick are present in this bag (recall that we do not delete parallel portal edges after contractions), all possible solutions restricted to the corresponding brick will be considered. Because the contracted brick vertices only appear in leaves of the dynamic programming tree, the rest of the dynamic programming algorithm can be carried out as in the standard case, considering the portal edges, whenever they occur again, as having zero length.

Analysis of the running time. As was shown in [BMK07b], the total time spent in the leaves of the dynamic programming is $O(4^\theta n)$. The rest of the dynamic programming takes time $O(2^{\mathrm{poly}(\omega)} n)$. The running time of the thinning algorithm is dominated by the contraction decomposition of Theorem 4.1 which is $O(n \log n)$ [CC07]. Hence, the total time is $O(2^{\mathrm{poly}(\omega)} n + n \log n)$ for the general case; particularly, this is singly exponential in ε^{-1} and g, as desired.

5. Constructing a Mortar Graph for Bounded-Genus Graphs

Let $G_0 = (V_0, E_0)$ be the input graph of genus g_0 and Q be the terminal set. In a first preprocessing step, we delete a number of unnecessary vertices and edges of G_0 to obtain a graph $G = (V, E)$ of genus $g \leq g_0$ that still contains every $(1+\varepsilon)$-approximate solution for terminal set Q for all $0 \leq \varepsilon \leq 1$ while fulfilling certain bounds on the length of shortest paths. In the next step, we find a *cut graph* CG of G that contains all terminals and whose length is bounded by a constant times OPT. We cut the graph open along CG, so that it becomes a planar graph with a simple cycle σ as boundary, where the length of σ is twice that of CG. See Figure 2 for an illustration. Afterwards, the remaining steps of building the mortar graph can be the same as in the planar case, by way of Theorem 3.2.

For an edge $e = vw$ in G_0, we let $\mathrm{dist}_{G_0}(r, e) = \min\{\mathrm{dist}_{G_0}(r, v), \mathrm{dist}_{G_0}(r, w)\} + \ell(e)$ and say that e is *at distance* $\mathrm{dist}_{G_0}(r, e)$ from r. If the root vertex represents a contracted graph H, we use the same terminology with respect to H.

5.1. Preprocessing the Input Graph

Our first step is to apply the following preprocessing procedure:

PREPROCESS(G_0, Q, ξ):
(1) Find a 2-approximate Steiner tree T_0 of G_0 and Q; contract T_0 to a vertex r.
(2) Find a shortest-path tree rooted at r.
(3) Delete all vertices v and edges e of G_0 with $\text{dist}_{G_0}(r, v), \text{dist}_{G_0}(r, e) > 2\xi\ell(T_0)$.

Any deleted vertex or edge is at distance $> 2\xi\ell(T_0) > 2\xi\,\text{OPT}$ from any terminal and hence, can not be part of a $(1+\varepsilon)$-approximation for any $0 \leq \varepsilon \leq 1$. We call the resulting graph $G = (V, E)$ and henceforth use G instead of G_0 in our algorithm. The preprocessing step can be accomplished in linear time: T_0 can be calculated with the recent improvement on Mehlhorn's algorithm [Meh88] by Tazari and Müller-Hannemann [TMH08], and the shortest path tree with Henzinger et al.'s algorithm [HKRS97]. Trivially, we have

Proposition 5.1. *Any vertex and any edge of G is at distance at most $4\xi\,\text{OPT}$ from T_0.*

5.2. Constructing the Cut Graph

Start again with T_0 contracted to a vertex r. We construct our cut graph as a *system of loops* rooted at r: iteratively find short non-separating cycles through r and cut the graph open along each cycle. Erickson and Whittlesey [EW05] showed that, for orientable surfaces, taking the *shortest* applicable cycle at each step results in the shortest system of loops through r. They suggest an implementation of their algorithm using the *tree-cotree decomposition*, introduced by Eppstein [Epp03], that runs in linear time on bounded-genus graphs. A tree-cotree decomposition of an embedded graph G is a triple $(T, T^\star, X)$, so that T is a spanning tree of G rooted at r, $T^\star$ is a spanning tree of the dual of $G \setminus T$, and X is the remaining set of edges (recall that we identify the edges of G and $G^\star$). Eppstein showed that the set of elementary cycles $\{\text{loop}(T, e) : e \in X\}$ is a cut graph of G where $\text{loop}(T, e)$ is the cycle formed by the paths in T from r to the endpoints of e plus the edge e. Eppstein's decomposition also works for non-orientable embeddings. As we only need to bound the length of our cut graph, we present a simpler algorithm:

PLANARIZE(G, R):
(1) Contract T_0 to become a single vertex r.
(2) Find a shortest paths tree SPT rooted at r.
(3) Uncontract r and set $T_1 = T_0 \cup SPT$. (T_1 is a spanning tree of G.)
(4) Find a spanning tree $T_1^\star$ in $(G \setminus T_1)^\star$. ($T_1^\star$ is a spanning tree of $G^\star$.)
(5) Let $X = E \setminus (T_1 \cup T_1^\star)$.
(6) Return $CG = T_0 \cup \{\text{loop}(T_1, e) : e \in X\}$.

Lemma 5.2. *The algorithm* PLANARIZE *returns a cut graph CG, so that cutting G open along CG results in a planar graph G_p with a distinguished face whose facial walk σ*

(1) is a simple cycle;
(2) contains all terminals (some terminals might appear more than once as multiple copies might be created during the cutting process);
(3) has length $\ell(\sigma) \leq 2(8\xi g + 2)\,\text{OPT}$.

The algorithm can be implemented in linear time.

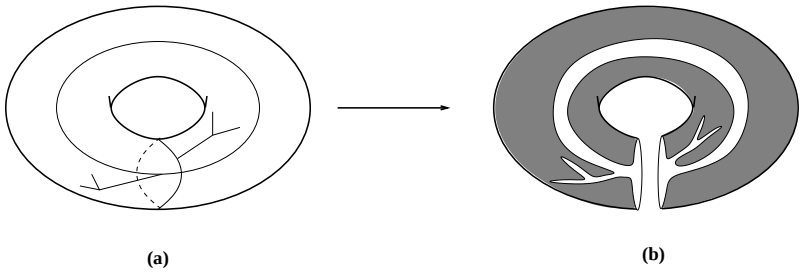


Figure 2: (a) a cut graph of a tree drawn on a torus; (b) the result of cutting the surface open along the cut graph: the shaded area is now homeomorphic to a disc and the white area is the new distinguished face of the planarized surface.

Proof. Clearly, $(T_1, T_1^\star, X)$ is tree-cotree decomposition of G and so, by Eppstein's lemma [Epp03], CG is a cut graph. By Euler's formula, we get that $|X| = g$, the Euler genus of G. Each edge $e = vw \in X$ completes a (non-surface-separating, not-necessarily simple) cycle as follows: a shortest path P_1 from T_0 to v, the edge e, a shortest path P_2 from w to T_0 and possibly a path P_3 in T_0. By Proposition 5.1, we know that e is at distance at most $4\xi\,\mathrm{OPT}$ from T_0 and so, P_1, P_2, and at least one of $P_1 \cup \{e\}$ and $P_2 \cup \{e\}$ have length at most $4\xi\,\mathrm{OPT}$. Hence, we have that $\ell(P_1 \cup \{e\} \cup P_2) \le 8\xi\,\mathrm{OPT}$. Because there are (exactly) g such cycles in CG, we get that

$$\ell(CG) \le g \cdot 8\xi\,\mathrm{OPT} + \ell(T_0) \le (8\xi g + 2)\,\mathrm{OPT}\,.$$

Because CG is a cut graph, it follows that it consists of a single facial walk σ'; this follows easily from Euler's formula, because CG has Euler genus g (because $G \setminus CG$ is planar), with some k vertices and $k + g - 1$ edges. So, σ' contains every edge of CG exactly twice (cf. [MT01]), i.e. $\ell(\sigma') = 2\ell(CG)$. Cutting the graph open along σ' results in a planar graph with a simple cycle $\sigma = \sigma'$ as its boundary, as desired (see Fig. 2).

As mentioned in the previous section, T_0 and SPT can be computed in linear time on bounded-genus graphs [HKRS97, TMH08]. $T_1^\star$ can be obtained, for example, by a simple breadth-first-search in the dual. The remaining steps can also easily be implemented in linear time. ∎

5.3. Proof of Theorem 3.3

We complete the construction of a mortar graph for genus-g embedded graphs.

Let G_p be the result of planarizing G as guaranteed by Lemma 5.2. G_p is a planar graph with boundary σ spanning Q and with length $\le 2(8\xi g + 2)\,\mathrm{OPT}$. Let MG be the mortar graph guaranteed by Theorem 3.2. Every edge of MG corresponds to an edge of G. Let MG' be the subgraph of G composed of edges corresponding to MG. Every face f of MG (other than σ) corresponds to a face f' of MG' and the interior of f' is homeomorphic to a disk on the surface in which G is embedded. It is easy to verify that MG' is a mortar graph of G satisfying the length bounds of Theorem 3.3.

Acknowledgements. The authors thank Jeff Erickson for pointing out that Theorem 4.1 can be implemented in $O(n \log n)$ time in both orientable and non-orientable surfaces.

References

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11–24, 1989.

[Aro03] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Mathematical Programming*, 97(1–2):43–69, 2003.

[BK08] G. Borradaile and P. N. Klein. The two-edge connectivity survivable network problem in planar graphs. In *ICALP '08: Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 485–501. Springer, 2008.

[BMK07] G. Borradaile, C. Mathieu, and P. N. Klein. A polynomial-time approximation scheme for Steiner tree in planar graphs. In *SODA '07: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1285–1294, 2007.

[BKP] G. Borradaile, P. N. Klein, and D. Pritchard. A polynomial-time approximation scheme for the survivable network problem in planar graphs. In preparation.

[BMK07b] G. Borradaile, C. Mathieu, and P. N. Klein. Steiner tree in planar graphs: An $O(n \log n)$ approximation scheme with singly exponential dependence on epsilon. In *WADS '07: Proceedings of the 10th Workshop on Algorithms and Data Structures*, volume 4619 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2007.

[CC07] S. Cabello and E. W. Chambers. Multiple source shortest paths in a genus g graph. In *SODA '07: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 89–97, Philadelphia, PA, USA, 2007.

[DHM07] E. D. Demaine, M. Hajiaghayi, and B. Mohar. Approximation algorithms via contraction decomposition. In *SODA '07: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 278–287, Philadelphia, PA, USA, 2007.

[EMAFV87] R. E. Erickson, C. L. Monma, and Jr. A. F. Veinott. Send-and-split method for minimum-concave-cost network flows. *Math. Oper. Res.*, 12(4):634–664, 1987.

[Epp03] D. Eppstein. Dynamic generators of topologically embedded graphs. In *SODA '03: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 599–608, Philadelphia, PA, USA, 2003.

[EW05] J. Erickson and K. Whittlesey. Greedy optimal homotopy and homology generators. In *SODA '05: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1038–1046, Philadelphia, PA, USA, 2005.

[HKRS97] M. Henzinger, P. N. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

[Kle06] P. N. Klein. A subset spanner for planar graphs, with application to subset TSP. In *STOC '06: Proceedings of the 38th ACM Symposium on Theory of Computing*, pages 749–756, 2006.

[KS90] E. Korach and N. Solel. Linear time algorithm for minimum weight Steiner tree in graphs with bounded treewidth. Manuscript, 1990.

[Meh88] K. Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Information Processing Letters*, 27:125–128, 1988.

[Moh99] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12(1):6–26, 1999.

[MT01] B. Mohar and C. Thomassen. *Graphs on Surfaces*. The John Hopkins University Press, 2001.

[RS86] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[TMH08] S. Tazari and M. Müller-Hannemann. Shortest paths in linear time on minor-closed graph classes, with an application to Steiner tree approximation. *Discrete Applied Mathematics*, in press, 2008. An extended abstract appeared in WG '08: Proceedings of the 34th Workshop on Graph Theoretic Concepts in Computer Science, LNCS 5344, pp. 360–371, Springer, 2008.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 183–194
www.stacs-conf.org

A POLYNOMIAL KERNEL FOR MULTICUT IN TREES

NICOLAS BOUSQUET [1] AND JEAN DALIGAULT [2] AND STÉPHAN THOMASSÉ [2] AND ANDERS YEO [3]

[1] ENS Cachan, 61, avenue du Président Wilson, 94235 Cachan cedex - France
E-mail address: nbousque@dptinfo.ens-cachan.fr

[2] Université Montpellier II - CNRS, LIRMM, 161 rue Ada 34392 Montpellier Cedex 5 - France
E-mail address: daligault@lirmm.fr

E-mail address: thomasse@lirmm.fr

[3] Royal Holloway, University of London, Egham Hill, EGHAM, TW20 0EX - UK
E-mail address: anders@cs.rhul.ac.uk

Abstract. The MULTICUT IN TREES problem consists in deciding, given a tree, a set of requests (i.e. paths in the tree) and an integer k, whether there exists a set of k edges cutting all the requests. This problem was shown to be FPT by Guo and Niedermeyer in [10]. They also provided an exponential kernel. They asked whether this problem has a polynomial kernel. This question was also raised by Fellows in [1].

We show that MULTICUT IN TREES has a polynomial kernel.

1. Introduction

An efficient way of dealing with NP-hard problems is to identify a parameter which contains its computational hardness. For instance, instead of asking for a minimum vertex cover in a graph - a classical NP-hard optimization question - one can ask for an algorithm which would decide, in $O(f(k).n^d)$ time for some fixed d, if a graph of size n has a vertex cover of size at most k. If such an algorithm exists, the problem is called *fixed-parameter tractable*, or FPT for short. An extensive litterature is devoted to FPT, the reader is invited to read [4], [7] and [12].

Kernelization is a natural way of proving that a problem is FPT. Formally, a *kernelization algorithm* receives as input an instance (I, k) of the parameterized problem, and outputs, in polynomial time in the size of the instance, another instance (I', k') such that

- $k' \leq k$,
- the size of I' only depends of k,
- the instances (I, k) and (I', k') are both true or both false.

Part of this research was supported by Alliance Project "Partitions de graphes orientés". Part of this research was supported by ANR Project GRAAL.

The reduced instance (I', k') is called a *kernel*. The existence of a kernelization algorithm clearly implies the FPT character of the problem since one can kernelize the instance, and then solve the reduced instance G', k' using brute force, hence giving an $O(f(k) + n^d)$ algorithm. A classical result asserts that being FPT is indeed equivalent to having kernelization. The drawback of this result is that the size of the reduced instance G' is not necessarily small with respect to k. A much more constrained condition is to be able to reduce to an instance of polynomial size in terms of k. Consequently, in the zoology of parameterized problems, the first distinction is done between three classes: W[1]-hard, FPT, polykernel.

A kernelization algorithm can be used as a preprocessing step to reduce the size of the instance before applying an algorithm. Being able to ensure that this kernel has actually polynomial size in k enhances the overall speed of the algorithm. See [11] for a recent review on kernalization.

The existence of a polynomial kernel can be a subtle issue. A recent result by Fernau et al [6] shows that Rooted k-Leaf Outbranching has a cubic kernel while k-Leaf Outbranching does not, unless polynomial hierarchy collapses to third level, using a breakthrough lower bound result by Bodlaender and al [5].

In the (unweighted) Multicut In Trees problem, we consider a tree T together with a set P of pairs of distinct nodes of T, called *requests*. Hence, a request can also be seen as a prescribed path joining these two nodes. We will often identify the request and its path. A *multicut* of (T, P) is a set S of edges of T which intersect every request in P, i.e. every path corresponding to a request contains an edge of S.

Problem 1.1. Multicut In Trees:

Input: A tree $T = (V, E)$, a set of requests P, an integer k.
Output: TRUE if there is a multicut of size at most k, otherwise FALSE.

Note that a more general presentation of this problem is to assign weights to edges, and ask for a multicut of minimal weight. Our technique does not seem to generalize to the weighted case.

This problem appears in network issues (routing, telecommunication, ...). See [3] for a survey on multicommodity flow problems and multicut problems. It was shown in [8] that Multicut In Trees is NP-complete, and its associated decision problem is MaxSNP-hard and has a factor-2 polynomial time approximation algorithm.

This problem is known to be FPT, see [9] or [10] for a branching algorithm and an exponential kernel. The existence of a polynomial kernel was asked in [1]. We verify that Multicut In Trees has indeed an $O(k^6)$ kernel. Our reduction is very much inspired from [9] and [10]. In the next section, we first illustrate our techniques when the tree T is a caterpillar. In Section 3 we extend the proof to general trees.

2. A polynomial kernel for caterpillars

A node of T which is not a leaf is an *internal node*. The *internal tree* of T is the tree restricted to its internal nodes. We say that T is a *caterpillar* if its internal tree is a path. We consider the restriction of the Multicut In Trees problem to caterpillars, as it contains the core of our proof in the general case.

Let us give some general definitions which will apply both for the caterpillar case and for the general case.

We say that two nodes x and y are *R-neighbors* if there exists a request xy. A leaf x and an internal node y are *quasi-R-neighbors* if there exists a request xy, or a request xz, where z is a leaf rooted at y. An internal node with no leaf attached to it is an *inner node*. If x is a leaf, we denote by $e(x)$ and call *the edge of* x the edge adjacent to x. A *group* of leaves is the set of leaves connected to the same internal node. A *group request* is a request xy where x and y belong to the same group. A leaf which is an endpoint of a group request is a *bad leaf*. A *leaf to leaf request* is a request between two leaves. An *internal request* is a request between two internal nodes. A request between an internal node and a leaf is a *mixed-request*. Two requests are *disjoint* if their edge sets are disjoint. Two requests x_1y_1 and x_2y_2 are *endpoint-disjoint* if x_1, y_1, x_2, y_2 are pairwise different.

The *internal path* of a request is the intersection between the path of the request and the internal tree. The *common factor* of two requests is the intersection of their paths. A request R_1 *dominates* a request R_2 if the internal path of R_1 contains the internal path of R_2.

Contracting an edge e in (T, P) means contracting e in T, and transforming each request of the form $(e_1, \ldots, e_t, e, e_{t+1}, \ldots, e_l)$ in P into $(e_1, \ldots, e_t, e_{t+1}, \ldots, e_l)$. *Deleting* an edge e means contracting e in T and removing every request containing e from P.

Two requests of length at least 2 from a given leaf x *have the same direction* if the second edge of their path starting at x is the same. Two requests from an internal node x *have the same direction* if the first edge of their paths (starting at x) is the same. All the requests from x *have the same direction* if they pairwise have the same direction.

In the following, our instance T is assumed to be a caterpillar. We call the two extremities of the internal path the *left end* and the *right end* of T. The path between a node x and the right (resp. left) end will be called *right* and *left* relatively to x.

Let T' be the internal tree of the caterpillar T. The following five sets partition T:

- The set I_1 of leaves of T'.
- The set I_2 of degree two nodes of T'.
- The set L_1 of leaves rooted at I_1.
- The set L'_2 of bad leaves rooted at I_2.
- The set L_2 of the other leaves rooted at I_2.

The *wingspan* W of a leaf x is the path between the closest quasi-R-neighbor on the right of x and the closest quasi-R-neighbor on the left of x (if no such neighbor exists, we take the father $f(x)$ of x by convention). The *size* of a wingspan is the number of L_2-leaves pending from it. The *subcaterpillar* of the wingspan W consists in W and the leaves rooted at W. The wingspan W *dominates* a request yz if both y and z belong to the subcaterpillar of W.

The usual way of exhibiting a kernel is to define a set of *reduction rules*. These rules should be *safe*, meaning that after applying a rule, the truth value of the problem on the instance does not change. Moreover the repeated application of the rules should take polynomial time. Finally, after iterating these rules on an instance, we want the *reduced instance* to be of polynomial size in k.

The reduction rules. We apply the following reduction rules to an instance:

(0) Unit Request: if a request R has length one, i.e. $R = e$ for some edge e of T, then we delete e and decrease k by one.

(1) Disjoint Requests: if there are $k+1$ disjoint requests in P, then we return a trivially false instance.
(2) Unique Direction: if all the requests starting at a leaf x have the same direction, then contract $e(x)$. If all the requests starting at an inner node x have the same direction, then contract the edge e adjacent to x which does not belong to any request starting at x.
(3) Inclusion: if a request R is included in another request R', then delete R' from the set of requests.
(4) Common Factor: let R be a request. If $k+1$ requests $R_1, \ldots, R_{k+1}$ different from R but intersecting R are such that for every $i \neq j$, the common factor of R_i and R_j is a subset of R, then delete R from the set of requests.
(5) Dominating Wingspan: if x is an L_2-leaf with a wingspan dominating at least $k+1$ endpoint-disjoint leaf to leaf or mixed requests, then contract $e(x)$.

Each iteration of the reduction consists in applying the first applicable rule, in the above order.

Lemma 2.1. *Rules Unit Request, Disjoint Requests, Unique Direction, Inclusion, Common Factor and Dominating Wingspan are safe.*

Proof. (0) Rule Unit Request is obvious.
(1) Rule Disjoint Requests is obvious.
(2) For Rule Unique Direction, assume first that all the requests from a leaf x have the same direction, and that a multicut contains $e(x)$. Let e' be the second common edge of all these paths. As e' cuts all the requests cut by $e(x)$, if $e(x)$ is in a solution S then $S \backslash \{e(x)\} \cup \{e'\}$ is also a solution. So we can contract $e(x)$. Now, assume that all the requests from an inner node x go to the right. If a solution S contains the edge e adjacent to x on the left then $S \backslash \{e\} \cup \{e'\}$, where e' is the right edge adjacent to x, is a solution since a request going through e also goes through e'.
(3) For Rule Inclusion, observe that an edge cutting R also cuts all the paths containing R.
(4) If there is a multicut of k edges, then one of these edges must intersect two requests among the $k+1$ mentioned in Rule Common Factor. This edge lies in the intersection of two paths, hence in R, so request R is cut in any multicut of $P \setminus \{R\}$.
(5) Let x be an L_2-leaf with a wingspan W dominating $k+1$ endpoint-disjoint requests. If a multicut of size k exists, it contains an edge e which cuts two of these requests. As the requests are endpoint-disjoint, their intersection is included in the internal tree, hence in W. Assume, for example, that e is on the left of the leaf x. Then all the requests from x which go to the left go through e, and moreover x has no group request. Thus, if a solution exists, there is a solution without $e(x)$, since $e(x)$ can be replaced by the edge e' which is on the right of the neighbor of x. ∎

Lemma 2.2. *Deciding whether a rule applies and applying it takes polynomial time.*

Proof. Denote by n the number of nodes in T and by r the number of requests, which is $O(n^2)$.

(0) The application of Rule Unit Request takes time $O(r)$.
(1) The maximum edge-disjoint paths problem in trees is polynomial, see [8], thus Rule Disjoint Requests is polynomial.

(2) Rule Unique Direction can be applied in time $O(rn^2)$.
(3) Rule Inclusion can be applied in time $O(r^2)$.
(4) For the running time of Rule Common Factor, consider a request R. Informally, we are looking for a large enough set of requests which intersect R, possibly leaving it at one or two places, such that the edges through which they leave are all distinct. More formally, let Z be the set of edges not in R but sharing a vertex with some edge in R. Let Y be the set of edges e in Z such that there exists a request starting at a node in R and going through e. We can assume without any loss that one request per such edge e is chosen. Let G be the graph which vertices are $Z - Y$ and which edges are the pairs (e, e') such that there exists a request going through both e and e'. There exist $k+1$ paths as in Rule Common Factor if and only if G has a matching of size at least $k + 1 - |Y|$. As the matching problem is polynomial, the application of Rule Common Factor takes polynomial time.
(5) Let W be a wingspan, let G be the graph which vertices are the leaves pending from W and where two leaves are adjacent if there is a request between them. There exist $k+1$ endpoint-disjoint requests dominated by W if and only if G has a matching of size $k+1$, thus Rule Dominating Wingspan is polynomial. ∎

Lemma 2.3. *The reduction process has a polynomial number of iterations.*

Proof. Each rule decreases the sum of the lengths of the requests, which is initially less than the number of requests times the number of nodes. ∎

In the following we consider an instance in which none of these rules can be applied, and prove that such a reduced instance has polynomial size in k.

Let us introduce two graphs theoretic lemmas which are used in our proof.

Lemma 2.4. *Let G be an undirected graph having m edges, of maximal positive degree Δ. Then G has a matching of size $\lfloor \frac{m}{2\Delta-1} \rfloor$.*

Proof. Such a matching can be obtained by a greedy algorithm, as taking an edge uv in the matching forbids the edges adjacent to u and those adjacent to v (there are at most $2\Delta - 1$ such edges, including uv). ∎

Lemma 2.5. *Let H be an undirected graph on n vertices, of maximal degree Δ. Then H has an independent set of size $\lfloor \frac{n}{\Delta+1} \rfloor$.*

Proof. Such an independent set can be obtained by a greedy algorithm, as taking a vertex u in the independent set forbids the vertices adjacent to u. ∎

Theorem 2.6. *The* Multicut In Caterpillars *problem has a kernel of size $O(k^5)$.*

The rest of this section is dedicated to the proof of the theorem.

Observation 2.7. A node has at most $k+1$ R-neighbors in each direction.

Proof. If a node x has $k+2$ R-neighbors in, say, the right direction, then Rule Common Factor applies to any longest right request of x. ∎

Claim 1. There are at most $2(k+1)(2k+1)-1$ bad leaves.

Proof. A bad leaf is connected to at most $k+1$ leaves of some given group, by Rule Common Factor. Let G be the undirected graph whose vertices are the bad leaves of T and where there is an edge between two leaves if there is a group request between them. The minimal degree in G is at least 1, and the maximal degree is at most $k+1$. If there are at least $2(k+1)(2k+1)$ bad leaves then there are at least $(k+1)(2k+1)$ edges in G. Thus by Lemma 2.4 there exist a matching of size $k+1$ which implies the existence of $k+1$ endpoint-disjoint (thus disjoint) group requests. In this case, Rule Disjoint Requests would apply. ∎

Claim 2. A wingspan has size at most $2(k+1)(4k+3)-1$.

Proof. Let W be a wingspan. As Rule Dominating Wingspan does not apply, W does not dominate $k+1$ endpoint-disjoint requests. Let W' be the set of leaves pending from W. Let G be the undirected graph which vertices are the leaves in W' and the nodes in W. For each leaf to leaf request zy such that z and y are in W', create an edge zy in G. For each mixed-request zy such that z is in W' and y in W, create an edge zy in G. Finding $k+1$ endpoint-disjoint requests is equivalent to finding a matching of size $k+1$ in G. The degree of a vertex u in G is at most $2k+2$ because there are at most $k+1$ requests in each direction for u in T (by Observation 2.7). Moreover, if u corresponds to a node of W', the degree of u is at least one. Indeed, since the wingspan of x is maximal, each L_2-leaf pending from W must have a request dominated by W.

If there are $2(k+1)(4k+3)$ L_2-leaves in W', then G contains at least $(k+1)(4k+3)$ edges, and so G has a matching of size $k+1$ by Lemma 2.4, which in turn means the existence of $k+1$ endpoint-disjoint requests. ∎

Claim 3. There are $O(k^3)$ L_2-leaves.

Proof. Let x be a L_2-leaf of wingspan W. By the previous claim, there are less than $2(k+1)(4k+3)$ leaves pending from W. At most $2(k+1)(4k+3)$ L_2-leaves not pending from W have wingspans intersecting W for each direction, as the furthest leaf (on the right) of wingspan intersecting W has a wingspan which dominates all other leaves of wingspan intersecting W from the right. Let H be the auxillary graph on L_2, where two L_2-leaves are adjacent if their wingspans intertsect. H has maximum degree less than $6(k+1)(4k+3)$ by the above discussion. By Lemma 2.5, if T has at least $6(k+1)(k+2)(4k+3)$ vertices, then H has a stable set of size $k+1$. Thus T would have $k+1$ disjoint wingspans, and thus $k+1$ disjoint requests, a contradiction.

Claim 4. There are $O(k^5)$ I_2-nodes.

Proof. By Claim 3, there are $O(k^3)$ I_2-nodes with leaves. Let us bound the number of inner nodes. Let I' be the set of inner nodes in T. Consider the graph G on the set of vertices I' where there is an edge xy if xy is a request in T.

Because of Rule Inclusion, each inner node has degree at most two in G (one in each direction). Thus G is a disjoint union of paths, called *request paths*. The length of a request path is at most k by Rule Disjoint Requests. A node with degree 1 in G is an *extremal inner node*.

Each extremal inner node must be an R-neighbor in T of a leaf or of an internal node with a leaf (otherwise it would be reduced by Rule Unique Direction). Denote by X the set of leaves and internal nodes with a leaf attached to it. Each node in X has $O(k)$ R-neighbors among the inner nodes, and $|X| = O(k^3)$, so there are $O(k^4)$ inner nodes with a neighbor in

X (in particular, at most $O(k^4)$ extremal inner nodes). Each extremal inner node belongs to a unique request path of size at most k. Moreover each inner node with no neighbor in X must belong to a request path. So there are $O(k^5)$ inner nodes in T. ∎

There are $O(k^3)$ leaves and $O(k^5)$ internal nodes in a reduced instance. Thus the MULTICUT IN CATERPILLARS problem has a kernel of size $O(k^5)$. ∎

3. General Trees

Should no confusion arise, we retain the terminology of the previous section.

Let (T, P, k) be an instance. Let T' be the tree obtained from T by deleting the leaves. We partition the set of nodes of T into the following seven sets:

- The set I_1 of leaves in T'.
- The set I_2 of degree 2 nodes in T'.
- The set I_3 of the other nodes in T'.
- The set L_1 of leaves rooted at I_1.
- The set L_2 of leaves rooted at I_2, endpoint of no group request.
- The set L'_2 of leaves rooted at I_2, endpoint of at least one group request.
- The set L_3 of leaves rooted at I_3.

We also denote by I the set of internal nodes of T, and by L the set of leaves of T.

We need a few technical definitions. A *caterpillar* of T is a maximal connected component of $T - I_3 - L_3$. The *backbone* of a caterpillar is the set of internal nodes of T in this caterpillar. A caterpillar C is *non-trivial* if the set of internal nodes in C seen as a caterpillar has size at least two. The *extremities* of a non-trivial caterpillar C are the two nodes of C which are I_2 or I_1-nodes of T and become I_1-nodes in C. A *minimal request* of a node x is a request having x as an endpoint and which internal path is minimal for inclusion among all internal paths of requests with x as an endpoint. If several requests have the same internal paths, we arbitrarily distinguish one as minimal and will not consider the others as minimal. If xy is a minimal request of x then y is called a *closest R-neighbor* of x.

Let x and y be nodes in T. If z lies on the path between x and y, or is a leaf rooted at the path between x and y, we say that z lies *toward* y from x (and we do not write "from x" should no confusion arise).

Assume x is an L_2-leaf of a caterpillar C (that is, an L_2-leaf of T which belongs to C). Let $f(x)$ be the node from which x is pending. Let $Gr(x)$ be the group of leaves pending from $f(x)$. Let $A(x)$ and $B(x)$ be the two connected components of $T - \{f(x)\} - Gr(x)$. Let $a(x)$ (resp. $b(x)$) be the extremity of C in $A(x)$ (resp. $B(x)$). If $A(x)$ (resp. $B(x)$) contains no extremity of C, that is if $f(x)$ is an extremity of C, then we define $a(x) = f(x)$ (resp. $b(x) = f(x)$). A *wingspan* W of x is formed by the restriction to internal nodes of the union of two requests between x and two of its closest R-neighbors lying respectively in $A(x)$ and $B(x)$. Observe that x can have several wingspans. The *subcaterpillar* of the wingspan W consists in W and the leaves rooted at W.

An L_2-leaf x *covers* a caterpillar C if either $x \notin C$ and there is a request starting at x and going through the whole backbone of C, or if $x \in C$ and there are two minimal requests starting at x which together cover the whole backbone of C.

We apply the following reduction rules to an instance: Rules (0), (1), (2), (3), and (4) are stated in the previous section. Rule Dominating Wingspan is split for convenience into two rules, one similar to the caterpillar case and a more general one, as follows:

(5a) Bidimensional Dominating Wingspan: if x is an L_2-leaf of a caterpillar C with a wingspan W such that $W \cap C$ dominates at least $k+1$ endpoint-disjoint requests, then we contract $e(x)$.

(5b) Generalized Dominating Wingspan: assume that x is an L_2-leaf of the caterpillar C, and that x covers C. Assume that for every closest neighbor z of x in $A(x)$, there exist $k+1$ endpoint-disjoint requests between a node lying toward $b(x)$ from x and a node toward z from $a(x)$. Then we contract $e(x)$.

Each iteration of the reduction consists in applying the first applicable rule, in the above order.

Lemma 3.1. *Rules (5a) and (5b) are safe.*

Proof. Safeness of Rule Bidimensional Dominating Wingspan follows from the safeness proof of Rule Dominating Wingspan in the previous section.

Assume Rule Generalized Dominating Wingspan can be applied to x. Let $z_1, \ldots, z_l$ be the closest R-neighbors of x in $A(x)$. For every $i \in \{1, \ldots, l\}$, because of the $k+1$ endpoint-disjoint requests mentionned in the rule, any k-multicut contains an edge in the path between z_i and $b(x)$. Assume that a k-multicut S contains an edge e'' between x and $b(x)$. Let e' be the edge adjacent to $e(x)$ in the path between x and $a(x)$. If S contains $e(x)$, then $S - \{e(x)\} \cup \{e'\}$ is also a k-multicut. Indeed, any request x, u with $u \in A(x)$ is cut by e', and any request x, v with $v \in B(x)$ is cut by e''. Assume now that a k-multicut S contains no edge between x and $b(x)$, then for every $i \in \{1, \ldots, l\}$, S must contain an edge e_i in the path between z_i and $f(x)$. Let e' be the edge adjacent to $e(x)$ in the path between x and $b(x)$. If S contains $e(x)$, then $S - \{e(x)\} \cup \{e'\}$ is a k-multicut. Indeed, any request x, u with $u \in A(x)$ is cut by an edge e_i, and any request x, v with $v \in B(x)$ is cut by e'. ∎

Proposition 3.2. *The repeated application of these rules on the instance until none can be applied takes polynomial time.*

Proof. The proof of the first five cases was made for general trees in the previous section. The polynomiality of Rule Bidimensional Dominating Wingspan follows from the proof of Rule Dominating Wingspan's polynomiality in the previous section. Deciding whether there exist $k+1$ endpoint-disjoint requests between prescribed areas can still be expressed as a matching problem as in Rule Dominating Wingspan's proof, so the application of Rule Generalized Dominating Wingspan also takes polynomial time. ∎

Theorem 3.3. *The number of nodes in a reduced instance is $O(k^6)$.*

The rest of this section is devoted to the proof of this theorem.

Claim 5. $|I_1| = O(k)$

Proof. There are at most k groups of leaves with a group request, by the $k+1$ disjoint requests rule. Every group of L_1-leaves has a group request, otherwise any leaf of this group would be deleted by Rule Unique Direction. Every I_1-node has at least one L_1-leaf pending from it, thus $|I_1| \leq k$. ∎

Claim 6. $|I_3| = O(k)$

Proof. In a tree, there are at most as many nodes of degree at least 3 as the number of leaves, so $|I_3| \leq |I_1| \leq k$. ∎

Claim 7. $|L_1| = O(k^2)$ and $|L'_2| = O(k^2)$

Proof. Each leaf in L_1 is a bad leaf by Rule Unique Direction, and each leaf in L'_2 is bad by definition. As in Claim 1 there are at most $2(k+1)(2k+1)-1$ bad leaves in T. Thus $|L_1 \cup L'_2| = O(k^2)$ ∎

We now show that:

- $|L_3| = O(k^4)$
- $|L_2| = O(k^4)$
- $|I_2| = O(k^6)$

Claim 8. The number of requests from a node x to a group of leaves is at most $k+1$.

Proof. Otherwise Rule Common Factor would apply to these requests. ∎

Claim 9. The number of requests from a node x to all the L_2-leaves in a given caterpillar C is at most $2k+2$ if $x \in C$ and $k+1$ if $x \notin C$.

Proof. Otherwise there would be at least $k+2$ requests sharing the same direction between x and leaves in this caterpillar, and Rule Common Factor would apply to these requests. ∎

Claim 10. There are at most $(2k+1)(k+2)-1$ requests between two groups of leaves.

Proof. Let G be the bipartite graph which vertices are the leaves of the two groups Y and Z, and where a leaf in Y and a leaf in Z are adjacent if there is a request between them. The maximum degree in G is at most $k+1$ by Claim 8, thus if there are $(2k+1)(k+2)$ requests between Y and Z, then by Lemma 2.4 there would be a matching of size $k+2$ in G. Thus there would be $k+2$ endpoint disjoint requests between Y and Z, and Rule Common Factor would apply. ∎

Claim 11. The number of requests between a group of leaves E and the nodes in a given caterpillar C is at most $2(2k+1)(k+2)-2$.

Proof. Assume by contradiction that there are at least $2(2k+1)(k+2)-1$ such requests. Let f be the node in which the leaves of E are rooted. If f belongs to C, then $C-f$ has two connected components. Among these two components, we select the component C' in which there is the largest number of requests from E. If f does not belong to C, then we let $C' = C$. There are at least $(2k+1)(k+2)$ requests between C' and E. Consider the undirected (bipartite) graph G which vertices are the leaves of E and the nodes of C', and where there is an edge between a leaf from E and node from C if there is a request between them. This graph has maximum degree $k+1$ by Rule Common Factor, thus by Lemma 2.4, G has a matching of size $k+2$. Thus there would be $k+2$ endpoint disjoint requests, and Rule Common Factor would apply to them. ∎

Claim 12. There are at most $2k-1$ caterpillars in T.

Proof. There are at most $2k$ nodes in $I_1 \cup I_3$. Let us call them *separating nodes*. Let r be one of these separating nodes. Let us consider r as the root of T. Each caterpillar is adjacent to exactly two separating nodes. Let us associate to each caterpillar of T its adjacent separating node further away from the root r. This mapping is a bijection, and no caterpillar is mapped on r, thus there are at most $2k-1$ caterpillars. ∎

Claim 13.
$|L_3| = O(k^4)$

Proof. We have that $|I_3| = O(k)$ by Claim 6. Let X be an L_3-group rooted in $y \in I_3$. Because of Rule Disjoint Requests, at most $(2k+1)(k+1)-1$ leaves in X are endpoints of group requests (by Lemma 2.4 on the usual auxilliary request graph on X). Each leaf of X must be the endpoint of at least one request, so let us count the maximal number of requests contributed by each type of nodes. By Claim 10, and as there are at most k groups of L_1-leaves and k groups of L_3-leaves, at most $k((2k+1)(k+2)-1)$ leaves of X have a request toward an L_1-leaf or an L_3-leaf. There are at most $2k-1$ caterpillars in T by Claim 12, and leaves in X have in total at most $2(2k+1)(k+2)-2$ R-neighbors in any caterpillar by Claim 11. Thus $O(k^3)$ leaves in X are endpoints of a request toward a caterpillar node, and I_3 nodes can contribute for at most $O(k^2)$ requests, so $|X| = O(k^3)$. This gives $|L_3| = O(k^4)$. ∎

Claim 14.
$|L_2| = O(k^4)$

Proof. Assume by contradiction that $|L_2| \geq 3(2k-1)(k+1)(k+1)(4k+3)$. Let C be a caterpillar of T containing the maximum number of L_2-leaves. By Claim 12, there are at most $2k-1$ caterpillars in T, thus C contains at least $3(k+1)(k+1)(4k+3)$ L_2-leaves.

Assume first that C is not covered. We obtain a contradiction as in the caterpillar case. Consider x to be the L_2-leaf having a wingspan which intersection $\tilde{W}$ with C has maximal size. Let C' be the subcaterpillar of backbone $\tilde{W}$. Then C' contains at least $(k+1)(4k+3)$ L_2-leaves, otherwise one would find $k+1$ disjoint wingspans by taking $\tilde{W}$, then a $\tilde{W}_1$ disjoint from $\tilde{W}$, then a $\tilde{W}_2$ disjoint from $\tilde{W}$ and $\tilde{W}_1$, ..., and finally a $\tilde{W}_k$ disjoint from $\tilde{W}, \tilde{W}_1, \ldots, \tilde{W}_{k-1}$, as in Claim 3. Note that the caterpillars $W, W_1, \ldots, W_k$ are disjoint, as their intersections $\tilde{W}, \tilde{W}_1, \ldots, \tilde{W}_k$ with C are disjoint and non-empty. Thus there would be $k+1$ disjoint requests, a contradiction. Since $\tilde{W}$ is maximal, each L_2-leaf y in C' is the endpoint of a request $r \subseteq C'$. The existence of $(k+1)(4k+3)$ such leaves means there are at least $k+1$ endpoint-disjoint requests dominated by $\tilde{W}$, by Lemma 2.4 applied to the usual auxiliary request graph G on the L_2-leaves of C' (note that the maximum degree of G is at most $2k+2$). Which means Rule (5a) should apply, a contradiction.

Assume now that C is covered by some L_2-leaf x. If more than $(k+1)(4k+3)$ L_2-leaves in C do not dominate C, then some wingspan of x dominates $(k+1)(4k+3)$ requests, and thus dominates at least $k+1$ endpoint-disjoint requests, by the usual application of Lemma 2.4. So Rule Bidimensional Dominating Wingspan should apply, a contradiction. So at least $3(k+1)(k+1)(4k+3) - (k+1)(4k+3)$ L_2-leaves in C cover C, let X be the set of these leaves. Let $d_1, \ldots, d_j$ be the I_1-nodes in $A(x)$. Note that $j \leq k$.

For such an I_1-node d_i and a leaf $x \in X$ having at least one quasi-R-neighbor lying toward d_i, let us denote by $rn(x,i)$ the closest quasi-R-neighbor of x toward d_i. Let $RN(i)$ be the set of all nodes $rn(x,i)$ for leaves $x \in X$ having at least one quasi-R-neighbor lying toward d_i. Note that the nodes of $RN(i)$ lie on the segment $[a(x), d_i]$. Denote by $x_1^i, \ldots, x_t^i$ the leaves in X having at least one quasi-R-neighbor lying toward d_i, ordered according to the distance between $a(x)$ and $rn(x,i)$, from closest to furthest. If $t \geq (k+1)(4k+3)$, denote by X_i the set $\{x_1^i, \ldots, x_{(k+1)(4k+3)}^i\}$.

When less than $(k+1)(4k+3)$ L_2-leaves in X have a quasi-R-neighbor toward d_i, mark d_i as invalid, and proceed. Note that at least one d_i must be valid, as $|X| > k(k+1)(4k+3)$.

Now we have a list of at most k sets (the sets X_i for d_i valid) of size $(k+1)(4k+3)$. The union X' of these is of size at most $k(k+1)(4k+3) < |X|$. Thus there exists an L_2-leaf z in $X - X'$. Consider the closest quasi-R-neighbor n_i of z toward a valid d_i. There are either $(k+1)(4k+3)$ L_2-leaves of X_i between z and $a(x)$ or $(k+1)(4k+3)$ L_2-leaves of X_i between z and $b(x)$. Thus there are $k+1$ endpoint-disjoint requests either between the subcaterpillars spanned by the segments $]z, a(x)[$ and $]a(x), n_i[$ or between the subcaterpillars spanned by the segments $]b(x), z[$ and $]a(x), n_i[$, by Lemma 2.4 on the usual auxiliary request graph. In the former case Rule Common Factor applies, in the latter Rule Generalized Dominating Wingspan applies. ∎

Claim 15.
$|I_2| = O(k^6)$

Proof. There are $O(k^4)$ internal nodes with leaves in T, by Claim 14. It remains to bound the cardinal of the set Z of inner nodes in I_2.

Let r be an I_1-node of T, we now consider r as the root of T. Let u be a node of Z. Let $C(u)$ be the caterpillar containing u, denote by $a(u)$ and $b(u)$ its extremities, with $b(u)$ an ancestor of $a(u)$ with respect to r. Let $A(u)$ be the connected component of $T - \{u\}$ containing $a(u)$. If the node u has an R-neighbor in $A(u)$, select such node $v(u)$. Note that u is on the path bewteen $v(u)$ and r. Thus, by Rule Inclusion, $v(u) \neq v(u')$ whenever $u \neq u'$. Let G be the graph with vertex set Z, and with edge set $\{(u, v(u)) | u \in Z\}$. This graph G is a disjoint union of paths. By Rule Disjoint Requests, paths in G have length at most k. Vertices u in G which have no R-neighbor in $A(u)$ must be adjacent in T to some node not in Z, by Rule Unique Direction. There are $O(k^4)$ nodes not in Z, each of which can have at most k R-neighbors in Z. Indeed, a vertex cannot have two different R-neighbors in the same direction, by Rule Inclusion. Thus there are $O(k^5)$ vertices u without R-neighbor in $A(u)$ in G, which gives that there are $O(k^6)$ vertices in G, which finally means that there are $O(k^6)$ inner nodes in T. □

This concludes the proof of the theorem. ∎

4. Conclusion

We have shown that the (unweighted) Multicut In Trees problem admits a polynomial kernel. This kernelization algorithm, or just some particular sequence using some of the reduction rules presented above, can be used as a preprocessing or in-processsing step in a practical algorithm.

This analysis might not be tight, so one can hope to improve this $O(k^6)$ bound retaining the same set of reduction rules. New reduction rules might be needed to decrease this bound even further.

Our technique does not seem to generalize to the weighted version of Multicut In Trees. Thus deciding whether the Weighted Multicut In Trees problem admits a polynomial kernel is still open.

It is not known whether the general Multicut in Graphs problem is FPT with respect to this parameter k, even for graphs of bounded treewidth. If it turned out to be true, then the question of the existence of a polynomial kernel for Multicut in Graphs would rise.

Among the most notorious open problems on polynomial kernelization stand Directed Feedback Vertex Set and Clique Cover. Directed Feedback Vertex Set consists in deciding whether a graph admits k vertices which removal makes the graph acyclic. This problem

was shown to be FPT in [2]. Clique Cover consists in deciding whether the edges of a graph can be covered by at most k cliques.

References

[1] H.L. Bodlaender, L. Cai, J. Chen, M.R. Fellows, J.A. Telle and D. Marx, Open problems in parameterized and exact computation. In H. L. Bodlaender and M. A. Langston, editors, *Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006*. Springer Verlag, Lecture Notes in Computer Science, vol. 4169, 2006.

[2] J. Chen,Y. Liu, S. Lu, B. O'Sullivan and I. Razgon. A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem. J. ACM, vol. 55, 1–19 (2008).

[3] M.-C. Costa, L. Letocart and F. Roupin: Minimal multicut and maximal integer multiflow: A survey, *European Journal of Operational Research* **162(1)** 55-69 (2005).

[4] R. G. Downey and M. R. Fellows. Parameterized Complexity. Springer, 1998.

[5] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels (extended abstract). Proceedings of the 34th International Colloquium on Automata, Languages and Programming, ICALP 2008, 563-574. Springer Verlag, Lecture Notes in Computer Science, vol. 5125, 2008.

[6] H. Fernau, F. Fomin, D. Lokshtanov, D. Raible, S. Saurabh and Y. Villanger. Kernel(s) for Problems With no Kernel: On Out-Trees With Many Leaves, accepted to STACS '08.

[7] J. Flum and M. Grohe. Parameterized Complexity Theory. Springer, 2006.

[8] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica* **18**, 3-20, 1997.

[9] J. Guo, Algorithm Design Techniques for Parameterized Graph Modification Problems, Feb. 2006, Ph.D. thesis, Institut fur Informatik, Friedrich-Schiller-Universitat, Jena, Germany.

[10] J. Guo and R. Niedermeier, Fixed-parameter tractability and data reduction for Multicut in Trees *Networks*, **46(3)**, 124–135, 2005, Wiley.

[11] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38:31-45, 2007.

[12] R. Niedermeier. Invitation to fixed-parameter algorithms. Oxford University Press, 2006.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 195–206
www.stacs-conf.org

ON LOCAL SYMMETRIES AND UNIVERSALITY IN CELLULAR AUTOMATA

LAURENT BOYER [1] AND GUILLAUME THEYSSIER [1]

[1] LAMA (CNRS, Université de Savoie),
Campus Scientifique, 73376 Le Bourget-du-lac cedex FRANCE
URL: `http://www.lama.univ-savoie.fr`
E-mail address, L. Boyer: `laurent.boyer@univ-savoie.fr`

E-mail address, G. Theyssier: `guillaume.theyssier@univ-savoie.fr`

Abstract. Cellular automata (CA) are dynamical systems defined by a finite local rule but they are studied for their global dynamics. They can exhibit a wide range of complex behaviours and a celebrated result is the existence of (intrinsically) universal CA, that is CA able to fully simulate any other CA. In this paper, we show that the asymptotic density of universal cellular automata is 1 in several families of CA defined by local symmetries. We extend results previously established for captive cellular automata in two significant ways. First, our results apply to well-known families of CA (e.g. the family of outer-totalistic CA containing the Game of Life) and, second, we obtain such density results with both increasing number of states and increasing neighbourhood. Moreover, thanks to universality-preserving encodings, we show that the universality problem remains undecidable in some of those families.

Introduction and definitions

The model of cellular automata (CA) is often chosen as a theoretical framework to study questions raised by the field of complex systems. Indeed, despite their formal simplicity, they exhibit a wide range of complexity attributes, from deterministic chaos behaviours (*e.g.* [3]) to undecidability in their very first dynamical properties (*e.g.* [2]). One of their most important feature is the existence of universal CA. Universality in CA is sometimes defined by an adaptation from the model of Turing machines and sequential calculus. But a stronger notion, intrinsic to the model of CA, has emerged in the literature [7]: a CA is *intrinsically universal* if it is able to fully simulate the behaviour of any other CA (even on infinite configurations).

Besides, when it comes to modelling [1] or experimental studies [10, 11], most works focus on some particular syntactical families (elementary CA, totalistic CA, etc), either to reduce the size of the rule space to explore, or to match hypothesis of the studied phenomenon at microscopic level (*e.g.* isotropy).

1998 ACM Subject Classification: F.1.1, G.2.1, F.4.3.
Key words and phrases: cellular automata, universality, asymptotic density.

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 195-206
http://drops.dagstuhl.de/opus/volltexte/2009/1836

In a word, CA are known for their general ability to produce complex global behaviours, but local rule considered in practice are often very constrained. This paper studies the link between syntactical restriction on CA local rules and typical global behaviours obtained. It establishes a probabilistic result: for various symmetry criterions over local rules, randomly choosing a local rule within the symmetric ones yields almost surely universal CA. Meanwhile, the universality problem is shown to remain undecidable even restricted to symmetric rules (for some of the symmetry criterions).

A family of CA defined by a simple syntactical constraint (namely *captive* CA) and containing almost only universal CA has already been proposed by one of the authors [9], but the present paper goes further. First, it generalises the probabilistic framework: the neighbourhood of CA is no longer fixed as it was needed in [9]. Second, it considers well-known families of CA (*e.g.* totalistic or outer-totalistic CA) and generalisations of them, namely *multiset* CA, which are meaningful for modelling (they are 'isotropic' CA).

After having recalled standard definitions about CA (end of this section), section 1 presents the families considered in this paper. Then, section 2 defines intrinsic universality and the simulation relation involved in that notion. Section 3 gives the probabilistic setting of the paper and establishes the main probabilistic results. Finally, section 4 is dedicated to existence proofs of universal CA in the families considered. Combined with probabilistic results, it proves that almost all CA are universal in those families.

Definitions and notations. In this paper, we adopt the setting of one-dimensional cellular automata. Formally, a CA is a 3-uple $\mathcal{A} = (n, k, \delta_{\mathcal{A}})$ where n and k are positive integers, respectively the size of the state set $\mathcal{Q}_n = \{1, \dots, n\}$ and of the neighbourhood $[\![-\lfloor\frac{k-1}{2}\rfloor; \lfloor\frac{k}{2}\rfloor]\!]$, $\delta_{\mathcal{A}} : \mathcal{Q}_n^k \to \mathcal{Q}_n$ is the *local transition function*.

A coloring of the lattice $\mathbb{Z}$ with states from $\mathcal{Q}_n$ (*i.e.* an element of $\mathcal{Q}_n^{\mathbb{Z}}$) is called a *configuration*. To $\mathcal{A}$ we associate a global function $G_{\mathcal{A}}$ acting on configurations by synchronous and uniform application of the local transition function. Formally, $G_{\mathcal{A}} : \mathcal{Q}_n^{\mathbb{Z}} \to \mathcal{Q}_n^{\mathbb{Z}}$ is defined by: $G_{\mathcal{A}}(x)_z = \delta_{\mathcal{A}}(x_{z-\lfloor\frac{k-1}{2}\rfloor}, \dots, x_{z+\lfloor\frac{k}{2}\rfloor})$ for all $x \in \mathcal{Q}_n^{\mathbb{Z}}$ and $z \in \mathbb{Z}$.

The local function $\delta_{\mathcal{A}}$ naturally extends to $\mathcal{Q}_n^*$, the set of finite words over alphabet $\mathcal{Q}_n$ (with $\delta_{\mathcal{A}}(u)$ being the empty word if $|u| < k$). For $p \in \mathbb{N}$, this function maps an element of $\mathcal{Q}_n^{p+k}$ to an element of $\mathcal{Q}_n^{p+1}$.

The size of $\mathcal{A} = (n, k, \delta_{\mathcal{A}})$ is the pair (n, k). The set of all CA is denoted by $\mathbf{CA}$, and the set of all CA of size (n, k) by $\mathbf{CA}_{n,k}$. Moreover for any set $\mathcal{F} \subseteq \mathbf{CA}$, $\mathcal{F}_{n,k}$ is defined by $\mathcal{F}_{n,k} = \mathcal{F} \cap \mathbf{CA}_{n,k}$. Formally a CA is a 3-uple but, to simplify notation, we sometimes consider that $\mathcal{F}_{n,k}$ is a set of local functions of type $\mathcal{Q}_n^k \to \mathcal{Q}_n$.

This paper will intensively use (finite) multisets. A multiset M of elements from a set E is denoted by $M = \{\!\{(e_1, n_1), \dots, (e_p, n_p)\}\!\}$ where a pair $(e_i, n_i) \in E \times \mathbb{N}$ denotes an element and its multiplicity. The cardinality of M is $|M| = \sum_i n_i$. The cardinality notation is the same for sets.

1. Families of CA with Local Symmetries

In this section, we define various families of CA characterised by some local symmetry. 'Symmetry' must be taken in a broad sense since it may concern various aspects of the local function. We first consider families where the local function does not depend on the exact configuration of the neighbourhood (a k-uple of states) but only on a limited amount of information extracted from this configuration.

MultiSet CA. Multiset cellular automata are cellular automata with a local rule invariant by permutation of neighbours. Equivalently, they are CA whose local function depends only on the multiset of states present in the neighbourhood. Formally, $\mathcal{A} \in \mathbf{CA}_{n,k}$ is *multiset*, denoted by $\mathcal{A} \in \mathbf{MS}_{n,k}$, if for any permutation π of $\{1 \dots k\}$, the local function $\delta_{\mathcal{A}}$ satisfies

$$\forall a_1, \dots, a_k \in \mathcal{Q}_n : \delta_{\mathcal{A}}(a_1, \dots, a_k) = \delta_{\mathcal{A}}(a_{\pi(1)}, \dots, a_{\pi(k)}).$$

Set CA. Set CA are a special case of multiset CA: they are CA whose local function depends only on the set of states present in the neighbourhood. Formally, $\mathcal{A} \in \mathbf{CA}_{n,k}$ with arity k is a *set CA*, denoted by $\mathcal{A} \in \mathbf{Set}_{n,k}$, if

$$\forall u, v \in \mathcal{Q}_n^k : \{u_1, \dots, u_k\} = \{v_1, \dots, v_k\} \Rightarrow \delta_{\mathcal{A}}(u) = \delta_{\mathcal{A}}(v).$$

Note that for fixed n, there is a constant N such that, for all k, $|\mathbf{Set}_{n,k}| \leq N$. Thus there is no hope that the asymptotic density of a non-trivial property for fixed n be 1 for family **Set**.

Totalistic CA. Totalistic CA are also a special case of Multiset CA: they are CA whose local functions depends only on the sum of the neighbouring states. Formally, $\mathcal{A} \in \mathbf{CA}_{n,k}$ k is *totalistic*, denoted by $\mathcal{A} \in \mathbf{Tot}_{n,k}$, if

$$\forall u, v \in \mathcal{Q}_n^k : \sum_{i=1}^{k} u_i = \sum_{i=1}^{k} v_i \Rightarrow \delta_{\mathcal{A}}(u) = \delta_{\mathcal{A}}(v).$$

Partial Symmetries. We can consider weaker forms of each family above, by excluding some neighbours from the 'symmetry' constraint and treating them as a full dependency in the local function. For instance, we define the set of *outer-multiset* CA as those with a local rule depending arbitrarily on a small central part of their neighbourhood and on the multiset of other neighbouring states. Formally, for any k', $0 \leq k' \leq k$, $\mathbf{O}_{k'}\mathbf{MS}_{n,k}$ is the set of CA with n states, arity k and such that for any permutation π of $\{1 \dots k - k'\}$ and any $a_1, \dots, a_{k-k'}, b_1, \dots, b_{k'} \in \mathcal{Q}_n$ we have:

$$\delta_{\mathcal{A}}(a_1, \dots, a_{\lfloor (k-k')/2 \rfloor}, b_1, \dots, b_{k'}, a_{\lfloor (k-k')/2 \rfloor + 1}, \dots, a_{k-k'}) = \delta_{\mathcal{A}}(a_{\pi(1)}, \dots, a_{\pi(\lfloor (k-k')/2 \rfloor)}, b_1, \dots, b_{k'}, a_{\pi(\lfloor (k-k')/2 \rfloor + 1)}, \dots, a_{\pi(k-k')}).$$

We define in a similar way *outer-totalistic* and *outer-set*, and denote them by $\mathbf{O}_{k'}\mathbf{Tot}_{n,k}$ and $\mathbf{O}_{k'}\mathbf{Set}_{n,k}$ respectively. Note that what is classically called *outer-totalistic* is exactly the family $\mathbf{O}_1\mathbf{Tot}_{n,k}$.

State symmetric CA. Families above are variations around the invariance by permutations of neighbours. State symmetric CA are CA with a local function invariant by permutation of the state set. Formally, a CA $\mathcal{A} \in \mathbf{CA}_{n,k}$ is *state symmetric*, denoted by $\mathcal{A} \in \mathbf{SS}_{n,k}$, if for any permutation π of $\mathcal{Q}_n$ we have:

$$\forall a_1, \dots, a_k : \delta_{\mathcal{A}}(a_1, \dots, a_k) = \pi^{-1}\big(\delta_{\mathcal{A}}(\pi(a_1), \dots, \pi(a_k))\big).$$

Note that we have a situation similar to the case of **Set**: for fixed k, there is a constant K such that, for all n, $|\mathbf{SS}_{n,k}| \leq K$. Thus their is no hope that the asymptotic density of a non-trivial property for fixed k be 1 in state symmetric CA.

Captive CA. Finally, we consider the family of captive CA already introduced in [8]: they are CA where the local function is constrained to produce only states already present in the neighbourhood. Formally, a CA $\mathcal{A} \in \mathbf{CA}_{n,k}$ is *captive*, denoted by $\mathcal{A} \in \mathbf{K}_{n,k}$, if:

$$\forall a_1, \dots, a_k : \delta_{\mathcal{A}}(a_1, \dots, a_k) \in \{a_1, \dots, a_k\}.$$

The following lemma shows a strong relationship between captive and state symmetric CA.

Lemma 1.1. *Let n and k be such that $1 \leq k \leq n-2$. Then we have $\boldsymbol{SS}_{n,k} \subseteq \boldsymbol{K}_{n,k}$.*

Combining symmetries. In the sequel, we will often consider intersections of two of the families above. Note that all intersections are generally non-trivial. However, for the case of $\mathbf{Tot}_{n,k}$ and $\mathbf{K}_{n,k}$, the intersection is empty as soon as there exists two k-uple of states with disjoint support but with the same sum, because the 'captive' constraint forces the two corresponding transitions to be different whereas the 'totalistic' constraint forces them to be equal. This happens for instance when $n \geq 3$ and k is even with k-uples $(1,3,1,3,\dots,1,3)$ and $(2,2,2,\dots,2)$.

2. Simulations and Universality

The property we are mostly interested in is intrinsic universality (see [7] for a survey on universality). To formalize it, we first define a notion of simulation.

A CA $\mathcal{A}$ is a *sub-automaton* of a CA $\mathcal{B}$, denoted $\mathcal{A} \sqsubseteq \mathcal{B}$, if there is an injective map φ from A to B such that $\overline{\varphi} \circ G_{\mathcal{A}} = G_{\mathcal{B}} \circ \overline{\varphi}$, where $\overline{\varphi} : A^{\mathbb{Z}} \to B^{\mathbb{Z}}$ denotes the uniform extension of φ to configurations. We sometimes write $\mathcal{A} \sqsubseteq_{\varphi} \mathcal{B}$ to make φ explicit. This definition is standard but yields to a very limited notion of simulation: a given CA can only admit a finite set of (non-isomorphic) CA as sub-automata. Therefore, following works of J. Mazoyer and I. Rapaport [4] and later N. Ollinger [5, 7], we will consider the following notion of simulation: a CA $\mathcal{A}$ simulates an AC $\mathcal{B}$ if some *rescaling* of $\mathcal{A}$ is a sub-automaton of some *rescaling* of $\mathcal{B}$. The ingredients of the rescalings are simple: packing cells into blocs, iterating the rule and composing with a translation (formally, we use shift CA σ_z, $z \in \mathbb{Z}$, whose global rule is given by $\sigma(c)_x = c_{x-z}$ for all $x \in \mathbb{Z}$). Formally, given any state set Q and any $m \geq 1$, we define the bijective packing map $b_m : Q^{\mathbb{Z}} \to \left(Q^m\right)^{\mathbb{Z}}$ by:

$$\forall z \in \mathbb{Z} : \big(b_m(c)\big)(z) = \big(c(mz), \dots, c(mz+m-1)\big)$$

for all $c \in Q^{\mathbb{Z}}$. The rescaling $\mathcal{A}^{<m,t,z>}$ of $\mathcal{A}$ by parameters m (packing), $t \geq 1$ (iterating) and $z \in \mathbb{Z}$ (shifting) is the CA of state set Q^m and global rule:

$$b_m \circ \sigma_z \circ G_{\mathcal{A}}^t \circ b_m^{-1}.$$

With these definitions, we say that $\mathcal{A}$ simulates $\mathcal{B}$, denoted $\mathcal{B} \preccurlyeq \mathcal{A}$, if there are rescaling parameters m_1, m_2, t_1, t_2, z_1 and z_2 such that $\mathcal{B}^{<m_1,t_1,z_1>} \sqsubseteq \mathcal{A}^{<m_2,t_2,z_2>}$. In the sequel, we will discuss *supports* of simulations, *i.e.* sets of configurations on which simulations occur. If $\mathcal{B}^{<m_1,t_1,z_1>} \sqsubseteq_{\varphi} \mathcal{A}^{<m_2,t_2,z_2>}$, the support of the simulation is the set of configuration of $\mathcal{A}$ defined by $b_{m_2}^{-1} \circ \overline{\varphi} \circ b_{m_1}(\mathcal{Q}_{\mathcal{B}}^{\mathbb{Z}})$. It is a subshift: a closed shift-invariant set of configurations. In the sequel we denote by $\mathcal{B} \preccurlyeq_X \mathcal{A}$ the fact that $\mathcal{A}$ simulates $\mathcal{B}$ on support X.

Once formalised the notion of simulation, we naturally get a notion of universality: CA able to simulate any other CA, denoted $\mathcal{A} \in \mathcal{U}$. This notion associated to $\preccurlyeq$ is called *intrinsic universality* in the literature (see [7]). Actually, an intrinsically universal CA $\mathcal{A}$

has the following stronger property (see [7, 5]): for all $\mathcal{B}$, there are rescaling parameters m, t and z such that $\mathcal{B} \sqsubseteq \mathcal{A}^{<m,t,z>}$.

3. Asymptotic Density and Monotone Properties

3.1. Asymptotic density

When considering a property $\mathcal{P}$ and a family $\mathcal{F}$ (two sets of CA), we can define the probability of $\mathcal{P}$ in $\mathcal{F}_{n,k}$ by $p_{n,k} = \frac{|\mathcal{F}_{n,k} \cap \mathcal{P}|}{|\mathcal{F}_{n,k}|}$. Our probabilistic framework consists in taking the limit of this probability $p_{n,k}$ when the "size" (n and/or k) of the automata grows toward infinity. In [9], only a particular case was considered: k fixed, and $n \to \infty$. The following definition consider all possible enumerations of 'size' through the notion of *path*.

Definition 3.1. A path is an injective function $\rho : \mathbb{N} \to \mathbb{N}^2$. When the limit exists, we define the asymptotic density of $\mathcal{P}$ in $\mathcal{F}$ following a path ρ by

$$d_{\rho,\mathcal{F}}(\mathcal{P}) = \lim_{x \to \infty} \frac{|\mathcal{F}_{\rho(x)} \cap \mathcal{P}|}{|\mathcal{F}_{\rho(x)}|}$$

The family of possible paths is huge and two different paths do not always define different densities.

We denote $\mathbb{N}_{c_0} = \mathbb{N} \setminus \{0, 1, \dots, c_0 - 1\}$. Since we consider asymptotics, we can restrain to paths $\rho : \mathbb{N} \to \mathbb{N}_{n_0} \times \mathbb{N}_{k_0}$ without loss of generality.

In the following, we will obtain limit densities of value 1, which justifies the use of non-cumulative density : in our case a density 1 following a given path implies a cumulative limit density 1 along this path.

3.2. Density of monotone properties among symmetric family

A property $\mathcal{P}$ is said to be *increasing* with respect to simulation if $\forall \mathcal{A} \in \mathcal{P}$, $\mathcal{A} \preccurlyeq \mathcal{B}$ implies $\mathcal{B} \in \mathcal{P}$. Decreasing properties are defined analogously. In this section we prove that monotone properties have density 0 or 1 among symmetric families introduced in section 1 following particular paths. More precisely, we are going to show that any non-trivial increasing property has density 1.

For any local function $f : \mathcal{Q}_n^k \to \mathcal{Q}_n$, for any set $E \subseteq \mathcal{Q}_n^k$, we denote by $f|_E$ the restriction of f to E. We also denote $\mathcal{F}_{n,k}|_E = \{f|_E : f \in \mathcal{F}_{n,k}\}$. Let $\{E_i\}_{i \in I}$ be a finite family of subsets of $\mathcal{Q}_n^k$ and denote $E = \cup_{i \in I} E_i$. We say that the family $\{E_i\}_{i \in I}$ is independent for $\mathcal{F}$ if the map

$$\psi : \mathcal{F}_{n,k} \ \to \ \mathcal{F}_{n,k}|_{\mathcal{Q}_n^k \setminus E} \times \prod_{i \in I} \mathcal{F}_{n,k}|_{E_i}$$

defined by $\psi(f) = (f|_{\mathcal{Q}_n^k \setminus E}, f|_{E_1}, \dots, f|_{E_i}, \dots)$ is a bijection (it is always injective).

By extension, we say that a collection of subshifts $\{X_i\}_{i \in I}$ is independent if the family $\{E(X_i)\}_{i \in I}$ is independent, where $E(X_i) \subseteq \mathcal{Q}_n^k$ is the set of words of length k occurring in X_i.

Let $\mathcal{S}_{\mathcal{A}_0} = \{\mathcal{A} \in \mathbf{CA} : \mathcal{A}_0 \preccurlyeq \mathcal{A}\}$ and $\mathcal{S}_{\mathcal{A}_0,X} = \{\mathcal{A} \in \mathbf{CA} : \mathcal{A}_0 \preccurlyeq_X \mathcal{A}\}$.

Lemma 3.2. *Let $\mathcal{F} \subseteq \boldsymbol{CA}$, and $\mathcal{A}_0 \in \mathcal{F}_{n_0,k_0}$ a given CA. For any size (n,k) ($n \geq n_0$, $k \geq k_0$) and any collection of subshifts $\{X_i\}_{i\in I}$, we denote $\alpha_i = \frac{|\mathcal{F}_{n,k} \cap \mathcal{S}_{\mathcal{A}_0,X_i}|}{|\mathcal{F}_{n,k}|}$ for all i. If $\{X_i\}_{i\in I}$ is independent for $\mathcal{F}$, then*

$$\frac{|\mathcal{F}_{n,k} \cap \mathcal{S}_{\mathcal{A}_0}|}{|\mathcal{F}_{n,k}|}. \geq 1 - \prod_{i\in\mathcal{I}} (1-\alpha_i)$$

Proof. We use the notations above. As the property $\mathcal{A}_0 \preccurlyeq_{X_i} \mathcal{A}$ is only determined by the restriction of $\mathcal{A}$ to $E(X_i)$, there exists $A_i \subseteq \mathcal{F}_{n,k}|_{E_i}$ such that $\psi(\mathcal{F}_{n,k} \cap \mathcal{S}_{\mathcal{A}_0,X_i}) = \mathcal{F}_{n,k}|_{\mathcal{Q}_n^k \setminus E} \times \mathcal{F}_{n,k}|_{E_1} \times \cdots \times \mathcal{F}_{n,k}|_{E_{i-1}} \times A_i \times \mathcal{F}_{n,k}|_{E_{i+1}} \cdots$. And as the family $\{E_i\}_{i\in I}$ is independent for $\mathcal{F}$, ψ is bijective and $\alpha_i = \frac{|A_i|}{|\mathcal{F}_{n,k}|_{E_i}|}$.

By definition of $\mathcal{S}_{\mathcal{A}_0}$ we have the following inclusion: $\bigcup_{i\in I}(\mathcal{F}_{n,k} \cap \mathcal{S}_{\mathcal{A}_0,X_i}) \subseteq (\mathcal{F}_{n,k} \cap \mathcal{S}_{\mathcal{A}_0})$. To conclude, it is enough to use the fact that ψ is bijective in order to express the size of these sets' complement in $\mathcal{F}_{n,k}$.

■

3.2.1. *Increasing n, fixed k.*

Proposition 3.3. *In the following, $\boldsymbol{E}$ is chosen among $\boldsymbol{CA}$, $\boldsymbol{MS}$, $\boldsymbol{Set}$, $\boldsymbol{O_{k'}Set}$, $\boldsymbol{O_{k'}MS}$. For any $\mathcal{A}_0 \in \boldsymbol{E} \cap \boldsymbol{K}_{n_0,k_0}$, for all ϵ, there exists n_{ϵ,k_0} such that if $n \geq n_{\epsilon,k_0}$*

$$\frac{|(\boldsymbol{E} \cap \boldsymbol{K}_{n,k_0}) \bigcap \mathcal{S}_{\mathcal{A}_0}|}{|\boldsymbol{E} \cap \boldsymbol{K}_{n,k_0}|} \geq 1 - \epsilon$$

Thus, any increasing property $\mathcal{P}$ such that $\exists \mathcal{A}_0 \in \mathbf{E} \cap \mathbf{K}_{n,k} \cap \mathcal{P}$ has density 1 in family $\mathbf{E} \cap \mathbf{K}$ for paths with fixed k. The case $\mathbf{E} = \mathbf{CA}$ was already proved in [9].

Proof. Let $\{X_i\}_{i\in[\![\,1;\lfloor \frac{n}{n_0} \rfloor\,]\!]}$ be a collection of fullshifts on disjoints alphabets of size n_0. They are independent for family $\mathbf{E}\cap\mathbf{K}$, whatever the choice of $\mathbf{E}$. Because of captivity constraint, the simulation happens on X_i with probability $\alpha_{i,n,k_0} \geq c_0 = 1/n_0^{n_0^{k_0}}$. We obtain by lemma 3.2 $\frac{|(\mathbf{E}\cap\mathbf{K}_{n,k_0}) \bigcap \mathcal{S}_{\mathcal{A}_0}|}{|\mathbf{E}\cap\mathbf{K}_{n,k_0}|} \geq 1 - (1-c_0)^{\lfloor \frac{n}{n_0} \rfloor}$.

■

3.2.2. *Increasing n, fixed k.* In the following, we use lemma 3.2, with an increasing number $l = O(k)$ of independent simulation subshifts, each providing the desired property for a constant fraction d_n of $\mathcal{F}_{n,k}$ (n is fixed). It gives $\frac{|\mathcal{F}_{n,k} \cap \mathcal{S}_{\mathcal{A}_0}|}{|\mathcal{F}_{n,k}|} \geq 1 - (1-d_n)^l$ and we obtain a limit density $d_{k,\mathcal{F}}(\mathcal{S}_{\mathcal{A}_0}) = 1$.

Multiset CA.

Proposition 3.4. *For all $\mathcal{A}_0 \in \boldsymbol{MS}_{n_0,k_0}$, for all $\epsilon > 0$, for all $n \geq n_0 + 2$, there exists k_ϵ such that for all $k > k_\epsilon$, $\frac{|\boldsymbol{MS}_{n,k} \cap \mathcal{S}_{\mathcal{A}_0}|}{|\boldsymbol{MS}_{n,k}|} > 1 - \epsilon$.*

Proof. We consider a multiset CA $\mathcal{A}_0 \in \mathbf{MS}_{n_0,k_0}$, a size $n \geq n_0 + 2k_0 + 4$, and a given $\epsilon > 0$. In order to clarify the construction we denote the 2 biggest states of $\mathcal{Q}_n$ by 0_0 and 1_0. For any size k, we define $l = \lfloor \frac{k-k_0}{k_0-1} \rfloor$ and $o = k - l.k_0$. And for any $j \in [\![k_0+1; l-k_0-1]\!]$, M_j is the word $M_j = 0_0^{l-j} \cdot 1_0^j$.

We define the simulating subshift X_j as the set of configurations alternating a state of $\mathcal{Q}_{n_0}$ and a pattern M_j. The family $\{X_j\}_j$ is independent for multiset CA. On every such subshift, the simulation will happen if the CA maintains the structure (eventually shifted) and computes steps of $\mathcal{A}_0$. Multisets corresponding to patterns of length k occurring in X_j are:

- $V_{j,\{\!\{(x_1,1),(x_2,1),\ldots,(x_{k_0},1)\}\!\}} =$
  $\{\!\{(0_0,(k_0-1).j+o),(1_0,(k_0-1).(l-j)),(x_1,1);(x_2,1),\ldots,(x_{k_0},1)\}\!\}$
- For $0 \leq s \leq o$, $W^0_{j,s,k_0-1} =$
  $\{\!\{(0_0,(k_0-1).j+o+1-s),(1_0,(k_0-1).(l-j)+s),(x_1,1),(x_2,1),\ldots,(x_{k_0-1},1)\}\!\}$
- $W^1_{j,k_0-1} = \{\!\{(0_0,(k_0-1).j),(1_0,(k_0-1).(l-j)+o+1),(x_1,1),(x_2,1),\ldots,(x_{k_0-1},1)\}\!\}$
- For $0 \leq s \leq o-1$, $W^{1'}_{j,s,k_0} =$
  $\{\!\{(0_0,(k_0-1).j+s),(1_0,(k_0-1).(l-j)+o-s),(x_1,1),(x_2,1),\ldots,(x_{k_0},1)\}\!\}$

$\mathcal{A}_0$ is simulated on support X_j if we have the following:

- $\delta_{\mathcal{A}}(V_{j,\{\!\{(x_1,1),(x_2,1),\ldots,(x_{k_0},1)\}\!\}}) = \delta_{\mathcal{A}_0}(\{\!\{(x_1,1),(x_2,1),...(x_{k_0},1)\}\!\})$
- $\delta_{\mathcal{A}}(W^0_{j,s,k_0-1}) = 0_0$ with $0 \leq s \leq o$
- $\delta_{\mathcal{A}}(W^1_{j,k_0-1}) = \delta_{\mathcal{A}}(W^{1'}_{j,s,k_0}) = 1_0$ with $0 \leq s \leq o-1$

The number of involved legal multiset transitions for a given subshift X_j is less than $(2.k_0+1).n_0^{k_0}$. Thus, the proportion of CA in $\mathbf{MS}_{n,k}$ simulating $\mathcal{A}_0$ on X_j is at least $1/n^{(2.k_0+1).n_0^{k_0}}$ which is constant with increasing k. And the number of such possible subshift is $l = O(k)$. We conclude with lemma 3.2 as explained before. ∎

Totalistic CA. We manage to make the multiset construction above to become totalistic. To do it, we define the mapping φ_j by: $\forall x \in \mathcal{Q}_{n_0}$, $\varphi_j(x) = (x(n_0+1)) \cdot 0_0^{l-j} \cdot 1_0^j$, with $0_0 = 0$ and $1_0 = n_0(n_0+1)+1$. The j-th subshift is defined as the smallest subshift containing $\left(\varphi_j(\mathcal{Q}_n^k)\right)^{\mathbb{Z}}$. The transitions are distinguishable by the number of 1_0, and the number of states smaller than $n_0(n_0+1)$ in any legal neighbourhood. The probability to simulate the original CA on the j-th subshift is constant, and the simulating subshifts are independent for totalistic CA. As the number of possible simulation increases, the limit probability for any CA to simulate a given CA is increasing to 1.

Outer-multiset CA. We still consider the same *possible* simulations of any multiset CA $\mathcal{A}_0 \in \mathbf{CA}_{n_0,k_0}$ by a CA $\mathbf{O}_{k'}\mathbf{MS}_{n,k}$.

As $\mathcal{A}$ is only partially multiset, the number of transitions involved in a simulation on one given subshift has increased: we have to consider the transitions with every possible central pattern of size k'. Using a precise account, we ensure that the number of transitions involved in one given simulation is bounded by $c^{k'}$ with c only depending on n_0 and k_0. And the number of possible subshifts for the simulation to happen is the same as in the totally multiset case: it is still given by $\lfloor k/2 \rfloor - 1$. We obtain $\frac{|\mathbf{O}_{k'}\mathbf{MS}_{n,k} \cap \mathcal{S}_{\mathcal{A}_0}|}{|\mathbf{O}_{k'}\mathbf{MS}_{n,k}|} > 1 - \left(1 - \frac{1}{(n_0+2)^{c^{k'}}}\right)^l$ with $l = O(k)$. To ensure that $d_{k,\mathbf{O}_{k'}\mathbf{MS}_{n,k}}(\mathcal{S}_{\mathcal{A}_0}) = 1$ it is enough to suppose that $k' = o(log(log(k))$.

3.2.3. *More general paths.*

Multiset captive CA. We prove a slightly more general result with the family of multiset captive CA **KMS** defined by $\mathbf{KMS} = \mathbf{K} \cap \mathbf{MS}$.

Proposition 3.5. *For any path* $\rho : \mathbb{N} \to \mathbb{N}^2$ *such that* the lower limit of $x \mapsto n = \pi_1(\rho(x))$ is infinite*, and for any* $\mathcal{A}_0 \in \boldsymbol{KMS}_{n_0,k_0}$*, for all* ϵ*, there exists* s_ϵ *such that if* $x > s_\epsilon$ *then*

$$\frac{|\mathcal{S}_{\mathcal{A}_0} \cap \boldsymbol{KMS}_{\rho(x)}|}{|\boldsymbol{KMS}_{\rho(x)}|} > 1 - \epsilon$$

Proof. The collection of subshifts, and the simulation behaviour are exactly the same as in the multiset case. If $\mathcal{A}_0$ is captive, each simulating transition is also captive. The number of involved transitions is the same as in the **MS** case: $(2.k_0 + 1).n_0^{k_0}$. But using the captivity constraint, the probability for the simulation on the j-th subshift to happen is also bounded by : $1/(2.k_0 + 1).n_0^{k_0}$. We use the fact that the number of possible simulations is still $O(k)$ to conclude using lemma 3.2.

■

Set captive CA.

Proposition 3.6. *For any path* $\rho : \mathbb{N} \to \mathbb{N}^2$ *such that* the lower limit of $x \to n = \pi_1(\rho(x))$ is infinite*, and for any* $\mathcal{A}_0 \in \boldsymbol{KSet}_{n_0,k_0}$*, for all* ϵ*, there exists* s_ϵ *such that if* $x > s_\epsilon$ *then*

$$\frac{|\mathcal{S}_{\mathcal{A}_0} \cap \boldsymbol{KSet}_{Path(x)}|}{|\boldsymbol{KSet}_{Path(x)}|} > 1 - \epsilon$$

Proof. Given $\mathcal{A}_0$, n, and k big enough, we denote the $2k_0 + 4$ first states of $\mathcal{Q}_n$ by 0_i and 1_i, $i \in [\![1; k_0 + 2]\!]$. The j-th subshift is the set of configurations alternating words $0_i^o 1_i^{l-o}$ (with $l = \lfloor \frac{k-k_0}{k_0-1} \rfloor$ and $o = k - l.k_0$) legally ordered and simulating states taken from $\Sigma_j = [\![2k_0 + 4 + j.n_0; 2k_0 + 4 + j.n_0 + n_0 - 1]\!]$. Legal set transitions for this subshift are

- $\{a_1, \ldots, a_{k_0}\} \cup \{\underline{0_i}, 1_i, \ldots, 0_{i+k-1}, 1_{i+k-1}, \underline{0_{i+k}}\} \to \delta_{\mathcal{A}_0}(\{a_1, \ldots, a_k\})$
- $\{a_1, \ldots, a_{k_0+e}\} \cup \{\underline{1_{i-1}}, 0_i, 1_i, \ldots, 0_{i+k-1}, 1_{i+k-1}, \underline{0_{i+k}}\} \to 1_{i+k/2}$ with $e \in \{0, -1\}$
- $\{a_1, \ldots, a_{k_0+e}\} \cup \{\underline{0_i}, 1_i, 0_{i+1}, 1_{i+1}, \ldots, 1_{i+k-1}, 0_{i+k}, \underline{1_{i+k}}\} \to 0_{i+k/2}$ with $e \in \{0, -1\}$

With indicies modulo $k + 2$, and $a_x \in \Sigma_j$ for all x. For all i those transitions may be identified by a set CA using the underlined state.

So we need $n_0 + 2.(k_0 + 2)$ different states to make the simulation on this subshift and the number of involved transitions is equal to $3.(k_0 + 2)$. Thus, because of captivity, the proportion p of CA in which one given simulation happens is constant when k, or n is increasing. And the family of the $\lfloor \frac{n-2(k_0+2)}{n_0} \rfloor$ possible simulation subshifts is independent. With lemma 3.2, we obtain the inequality $\frac{|\mathcal{S}_{\mathcal{A}_0} \cap \mathbf{KSet}_{n,k}|}{|\mathbf{KSet}_{n,k}|} > 1-(1-p)^{\lfloor \frac{n-2(k_0+2)}{n_0} \rfloor}$. We conclude the proof using the hypothesis on the path, $\underline{lim}_{x\to\infty} n = \underline{lim}_{x\to\infty} \pi_1(x) = \infty$. ■

4. Encodings

In the following we prove that there exists universal cellular automata in most of the families defined above. This is an important step considering the fact that some well known locally defined family, such as LR-permutative CA, do not contain any universal CA (because intrinsic universality implies non-surjectivity, see [5]). In fact, for every given family $\mathcal{F}$, we introduce an encoding map $\varphi_{\mathcal{F}} : \mathbf{CA} \to \mathcal{F}$ such that for any $\mathcal{A}$, its corresponding encoded version $\varphi_{\mathcal{F}}(\mathcal{A})$ verifies $\mathcal{A} \preccurlyeq \varphi_{\mathcal{F}}(\mathcal{A})$. The existence of a universal CA in $\mathcal{F}$ follows by application of the encoding to any universal CA. Moreover, in some cases, we obtain a stronger result: the encoded CA is universal if and only if the original CA is universal.

Set CA. Given a CA $\mathcal{A} \in \mathbf{CA}_{n,k}$ of state set $\mathcal{Q}_n$, we construct $\Psi(\mathcal{A}) \in \mathbf{Set}$ with state set $Q = \mathcal{Q}_n \times \{0, \dots, k+1\} \cup \{\#\}$ of size $n.(k+2)+1$.

A configuration $c \in Q^{\mathbb{Z}}$ is said *legal* if $c(z) \neq \#$ for all z and if the projection of c on the second component of states (which is well-defined) is periodic of period $1 \cdot 2 \cdots (k+2)$. Thus, for any legal configuration c and any position z, the set of states of cells which are neighbours of z is of the form:

$$E_i(a_1, \dots, a_k) = \{(a_1, i), (a_2, i+1 \ mod\ k+2), ..., (a_k, i+k-1 \ mod\ k+2)\}$$

for some $i \in \{1, \dots, k+2\}$ (with $a_j \in \mathcal{Q}_n$ for all j). $\Psi(\mathcal{A})$ is defined by the local rule f as follows:

$$f(x_1, \dots, x_k) = \begin{cases} \big(\delta_{\mathcal{A}}(a_1, \dots, a_k), i + \lfloor k/2 \rfloor \ mod\ k+2\big) & \text{if } \{x_1, \dots, x_k\} = E_i(a_1, \dots, a_k), \\ \# & \text{else.} \end{cases}$$

By construction, we have $\Psi(\mathcal{A}) \in \mathbf{Set}$. Moreover the encoding preserves universality. As a direct corollary, we get the undecidability of universality in family **Set** (universality was proven undecidable in the general case in [6]).

Theorem 4.1. *The encoding $\Psi : \boldsymbol{CA} \to \boldsymbol{Set}$ satisfies the following:*

(1) *$\mathcal{A} \preccurlyeq \Psi(\mathcal{A})$ for all $\mathcal{A}$,*
(2) *$\mathcal{A}$ is universal if and only if $\Psi(\mathcal{A})$ is universal.*

Captive set CA. We denote by **KSet** the intersection $\mathbf{K} \cap \mathbf{Set}$. The previous construction does not generally produce captive CA (even if the original CA is captive). We now describe a new encoding which produces only CA belonging to **KSet**. It could have been used to prove the existence of universal set CA, but we have no proof that it satisfies the second assertion of theorem 4.1 (hence the usefulness of previous construction).

The new mapping $\varphi : \mathbf{CA} \to \mathbf{KSet}$ is an adaptation of Ψ. We keep the idea of states being a cartesian product of the original alphabet $\mathcal{Q}_n$ and a family of labels which is in this case $\{0, ..., 2k-2\}$. But, in order to have every transition satisfying the captive constraint, we introduce 'libraries' of states placed regularly in legal configurations: between two computing cells, we place the i-th library for some i, denoted by $\mathbf{L}_i$, which contains the n states $\{(x, i)\}_{x \in \mathcal{Q}_n}$. For technical reasons, it also contains special states $(\#, i)$ and $(\#', i)$, and it is ordered as follows: $\mathbf{L}_i = (\#, i) \cdot (1, i) \cdot (2, i) \cdots (n, i) \cdot (\#', i)$. Thus, $\varphi(\mathcal{A})$ has state set $Q = \{0, \dots, 2k-2\} \times (\mathcal{Q}_n \cup \{\#, \#'\})$.

The simulation of $\mathcal{A}$ by $\varphi(\mathcal{A})$ takes place on 'legal' configurations defined by an alternation of an isolated state of label i, and a library of type $k+i$, precisely:

$$\cdots\ (q_1,i)\ \cdot\ \mathbf{L}_{k+i \bmod 2k-1}\ \cdot\ (q_2, i+1 \bmod 2k-1)\ \cdot\ \mathbf{L}_{k+i+1 \bmod 2k-1}\cdots$$

Those legal configurations will be maintained in one-to-one correspondence with configurations of $\mathcal{A}$, successive isolated states between libraries corresponding to successive states from $\mathcal{A}$. However, this time, the simulation of 1 step of $\mathcal{A}$ will use 2 steps of $\varphi(\mathcal{A})$ and only even time steps of $\varphi(\mathcal{A})$ (including time 0) will produce legal configurations. For odd time steps, we introduce 'intermediate' configurations defined by an alternation of an isolated state of label i, and a library of type $r+i$, precisely:

$$\cdots\ (q_1,i)\ \cdot\ \mathbf{L}_{r+i \bmod 2k-1}\ \cdot\ (q_2, i+1 \bmod 2k-1)\ \cdot\ \mathbf{L}_{r+i+1 \bmod 2k-1}\cdots$$

where $r = \lfloor k/2 \rfloor$ is the radius of $\mathcal{A}$.

To describe the local rule of $\varphi(\mathcal{A})$, we introduce the following sets:

- $V_i(a_1,\ldots,a_k) = \{(a_1,i),(a_2,i+1 \bmod 2k-1)\ldots(a_k,i+k-1 \bmod 2k-1)\}$;
- L_i is the set of states present in the word $\mathbf{L}_i$;
- $B_{i,x} = \{(\#,i),(1,i),...,(b-1,i)\}$ is the set of states in the prefix of of $\mathbf{L}_i$ of length x;
- $E_{i,x} = \{(e,i),...,(n,i),(\#',i)\}$ is the set of states in the suffix of $\mathbf{L}_i$ of length $n-x+1$.

$\varphi(\mathcal{A})$ has arity $k' = k+(k-1)\cdot(n+2)$ and, on legal configurations, the set of states seen in a neighbourhood has one of the following types:

T1: $V_i(a_1,\ldots,a_k) \cup L_{i+k \ mod\ 2k-1} \cup \ldots \cup L_{i+2k-2 \ mod\ 2k-1}$;
T2: $V_i(a_1,\ldots,a_k) \cup E_{i+k-1 \ mod\ 2k-1,x} \cup L_{i+k \ mod\ 2k-1} \cup \ldots$
$\ldots \cup L_{i+2k-3 \ mod\ 2k-1} \cup B_{i+2k-2 \ mod\ 2k-1,x}$.

On intermediate configurations, the set of states seen in a neighbourhood has one of the following types:

T3: $V_i(a_1,\ldots,a_k) \cup L_{i-r \ mod\ 2k-1} \cup \ldots \cup L_{i-r+k-2 \ mod\ 2k-1}$;
T4: $V_i(a_1,\ldots,a_k) \cup E_{i-r-1 \ mod\ 2k-2,x} \cup L_{i-r \ mod\ 2k-1} \cup \ldots$
$\ldots \cup L_{i-r+k-3 \ mod\ 2k-1} \cup B_{i-r+k-2 \ mod\ 2k-1,x}$.

An important point is that the 4 types are disjoint: it is obvious that each of T1 and T3 is disjoint from each of T2 and T4, and the overall disjointness follows from the fact that sets of type T3 and T4 have less elements than T1 and T2 since set L_i are disjoint but

$$V_i(a_1,\ldots,a_k) \cap L_j \neq \emptyset \iff i \leq j \leq i+k-1$$

Using notations above, the behaviour of $\varphi(\mathcal{A})$ is defined by 4 kinds of transitions according to the kind of neighbourhood seen:

T1: $\rightarrow (\delta_{\mathcal{A}}(a_1,...a_k), i+2k-2 \bmod 2k-1)$
T2: $\rightarrow (x, i+k-1 \bmod 2k-1)$
T3: $\rightarrow (a_{1+r}, i+r \bmod 2k-1)$
T4: $\rightarrow (x, i+r)$

The crucial point for transition of type T3 to be well-defined is that a_{1+r} can be unambiguously determined given that the libraries present have labels ranging from $i-r-1$ to $i-r+k-2 = i+r-1$ whereas a_{1+r} is associated to label $i+r$ in $V_i(a_1,\ldots,a_k)$ (everything is taken modulo $2k-1$).

Intuitively, type T1 corresponds to simulation of transitions of $\mathcal{A}$ and the three other types are devoted to the modification of label of isolated states or the displacement of libraries according to the following scheme:

- At even steps, transitions of type T1 apply the local rule $\delta_{\mathcal{A}}$, but the result receive a label j such that $\mathbf{L}_j$ is present in the neighbourhood to satisfy captivity constraint; meanwhile, transitions of type T2 just shift the libraries.
- At odd steps, the difference of labels between libraries and isolated states is wrong; to come back to a legal configuration, transitions of type T3 leave isolated states unchanged while transitions of type T4 shift the libraries.

To completely define $\varphi(\mathcal{A})$, we fix some ordering on Q and specify that, when the set E of neighbours doesn't correspond to any of the 4 types above, the local rule of $\varphi(\mathcal{A})$ simply chooses the greatest state in E. With that definition, $\varphi(\mathcal{A})$ always belong to **KSet**, because it depends only on the set of states in the neighbourhood, and because each transition produces a state already present in the neighbourhood (either the neighbourhood contains L_i for the right value of i, or the local rule simply chooses the greatest element).

Theorem 4.2. *For any $\mathcal{A}$, we have $\mathcal{A} \preccurlyeq \varphi(\mathcal{A})$. Therefore families **MS**, **KMS**, **Set** and **KSet** contain universal CA.*

The construction above corresponds to the strongest symmetry constraint (captive set CA), put aside totalistic CA. The existence of (intrinsically) universal totalistic CA is proven in [5]. The case of outer-totalistic CA follows by inclusion.

5. Universality Everywhere

Gathering the density results of section 3 and the existential results for universality in section 4, we obtain an asymptotic density 1 for universality in the following classes.

Family $\mathcal{F}$	**Condition on the path ρ**	**Comments**
Captive CA	$\exists k_0$ s.t. $\rho(x) = (x, k_0)$	Already in [9]
Multiset CA	$\exists n_0$ s.t. $\rho(x) = (n_0, x)$	
k'-outer-multiset	$\exists n_0$ s.t. $\rho(x) = (n_0, x)$	$k' = o(\log(\log k))$
Totalistic CA	$\exists n_0$ s.t. $\rho(x) = (n_0, x)$	
k'-outer-totalistic	$\exists n_0$ s.t. $\rho(x) = (n_0, x)$	$k' = o(\log(\log k))$
Set captive CA	$\underline{\lim}_{x\to\infty} \pi_1(\rho(x)) = +\infty$	
Multiset captive CA	$\underline{\lim}_{x\to\infty} \pi_1(\rho(x)) = +\infty$	

6. Open Problems and Future Work

As summarised in the previous section, our work establishes that universality has asymptotic density 1 along path ρ in several families defined by local symmetries, provided ρ verifies some hypothesis depending on the family considered.

Notably, we leave open the question of the density of universality in the following cases:

- increasing state set for families **MS**, **Set**, **Tot** (and outer-versions),
- increasing neighbourhood for family **K**.

We have no result (and no intuition) concerning the case of the whole set of CA either. A possible progress on that topic could be to reduce the density problem of a family $\mathcal{F}_1$ to the density problem of a family $\mathcal{F}_2$, *i.e.* to show that the densities (if they exist) in the two families are equal up to non-trivial multiplicative constants.

Another perspective, especially for multiset CA (or sub-families **Set** and **Tot**), is to extend our result to higher dimensions or even to more general lattice of cells. Indeed, the symmetry involved here implies isotropy which is an often required property in modelling.

Finally, it remains to study typical dynamics obtained in each family from random initial configuration. Experiments suggest that self-organisation in those families is far more frequent than in CA in general.

References

[1] Bastien Chopard and Michel Droz. *Cellular automata modeling of physical systems*. Collection Aléa-Saclay: Monographs and Texts in Statistical Physics. Cambridge University Press, Cambridge, 1998.

[2] J. Kari. Reversibility and Surjectivity Problems of Cellular Automata. *Journal of Computer and System Sciences*, 48(1):149–182, 1994.

[3] P. Kůrka. Languages, equicontinuity and attractors in cellular automata. *Ergodic Theory and Dynamical Systems*, 17:417–433, 1997.

[4] J. Mazoyer and I. Rapaport. Inducing an Order on Cellular Automata by a Grouping Operation. In *Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science, 1998.

[5] N. Ollinger. *Automates Cellulaires : structures*. PhD thesis, École Normale Supérieure de Lyon, dcembre 2002.

[6] N. Ollinger. The intrinsic universality problem of one-dimensional cellular automata. In *Symposium on Theoretical Aspects of Computer Science*, pages 632–641. Lecture Notes in Computer Science, 2003.

[7] Nicolas Ollinger. Universalities in cellular automata: a (short) survey. In B. Durand, editor, *Symposium on Cellular Automata Journées Automates Cellulaires (JAC'08)*, pages 102–118. MCCME Publishing House, Moscow, 2008.

[8] G. Theyssier. Captive cellular automata. In *Mathematical Foundations of Computer Science*, pages 427–438. Lecture Notes in Computer Science, 2004.

[9] G. Theyssier. How common can be universality for cellular automata? In *Annual Symposium on Theoretical Aspects of Computer Science*, 2005.

[10] Wolfram. Statistical mechanics of cellular automata. *Review of Modern Physics*, 55, 1983.

[11] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 207–218
www.stacs-conf.org

QUALITATIVE REACHABILITY IN STOCHASTIC BPA GAMES

TOMÁŠ BRÁZDIL [1] AND VÁCLAV BROŽEK [2] AND ANTONÍN KUČERA [2] AND JAN OBDRŽÁLEK [2]

[1] Institut für Informatik, TU München, Boltzmannstr. 3, 85748 Munich, Germany.
E-mail address: brazdil@fi.muni.cz
URL: http://www7.in.tum.de/

[2] Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic.
E-mail address: brozek@fi.muni.cz
E-mail address: kucera@fi.muni.cz
E-mail address: obdrzalek@fi.muni.cz
URL: http://www.fi.muni.cz/

Abstract. We consider a class of infinite-state stochastic games generated by stateless pushdown automata (or, equivalently, 1-exit recursive state machines), where the winning objective is specified by a regular set of target configurations and a qualitative probability constraint '>0' or '=1'. The goal of one player is to maximize the probability of reaching the target set so that the constraint is satisfied, while the other player aims at the opposite. We show that the winner in such games can be determined in **NP** ∩ **co-NP**. Further, we prove that the winning regions for both players are regular, and we design algorithms which compute the associated finite-state automata. Finally, we show that winning strategies can be synthesized effectively.

1. Introduction

Stochastic games are a formal model for discrete systems where the behavior in each state is either controllable, adversarial, or stochastic. Formally, a stochastic game is a directed graph G with a denumerable set of vertices V which are split into three disjoint subsets $V_\Box$, $V_\Diamond$, and $V_\bigcirc$. For every $v \in V_\bigcirc$, there is a fixed probability distribution over the outgoing edges of v. We also require that the set of outgoing edges of every vertex is nonempty. The game is initiated by putting a token on some vertex. The token is then moved from vertex to vertex by two players, $\Box$ and $\Diamond$, who choose the next move in the vertices of $V_\Box$ and $V_\Diamond$, respectively. In the vertices of $V_\bigcirc$, the outgoing edges are chosen according to the associated fixed probability distribution. A *quantitative winning objective* is specified by some Borel set W of infinite paths in G and a probability constraint $\rhd\varrho$, where $\rhd \in \{>, \geq\}$ is a comparison and $\varrho \in [0, 1]$. An important subclass of quantitative winning objectives are *qualitative winning objectives* where the constant ϱ must be either 0 or 1. The goal of player $\Box$ is

1998 ACM Subject Classification: G.3 PROBABILITY AND STATISTICS, F.1 COMPUTATION BY ABSTRACT DEVICES.

Key words and phrases: Stochastic games, reachability, pushdown automata.

All authors are supported by the research center Institute for Theoretical Computer Science (ITI), project No. 1M0545. T. Brázdil is also supported by the Alexander von Humboldt Foundation, and J. Obdržálek by grant GAČR 201/08/0308.

to maximize the probability of all runs that stay in W so that it is $\triangleright$-related to ϱ, while player $\diamond$ aims at the opposite. A *strategy* specifies how a player should play. In general, a strategy may or may not depend on the history of a play (we say that a strategy is *history-dependent (H)* or *memoryless (M)*), and the edges may be chosen deterministically or randomly (*deterministic (D)* and *randomized (R)* strategies). In the case of randomized strategies, a player chooses a probability distribution on the set of outgoing edges. Note that deterministic strategies can be seen as restricted randomized strategies, where one of the outgoing edges has probability 1. Each pair of strategies (σ, π) for players $\Box$ and $\diamond$ determines a *play*, i.e., a unique Markov chain obtained from G by applying the strategies σ and π in the natural way. The *outcome* of a play initiated in v is the probability of all runs initiated in v that are in the set W, denoted $\mathcal{P}_v^{\sigma,\pi}(W)$. We say that a play is $(\triangleright\varrho)$-won by player $\Box$ if its outcome is $\triangleright$-related to ϱ; otherwise, the play is $(\triangleright\varrho)$-won by player $\diamond$. A strategy of player $\Box$ (or player $\diamond$) is $(\triangleright\varrho)$-*winning* if for every strategy of the other player, the corresponding play is $(\triangleright\varrho)$-won by player $\Box$ (or by player $\diamond$, respectively). A natural question is whether one of the two players always has a $(\triangleright\varrho)$-winning strategy, i.e., whether the game is *determined*. The answer is somewhat subtle. A celebrated result of Martin [18] (see also [17]) implies that stochastic games with Borel winning conditions are *weakly determined*, i.e., each vertex v has a *value* given by

$$val(v) \quad = \quad \sup_{\sigma} \inf_{\pi} \mathcal{P}_v^{\sigma,\pi}(W) \quad = \quad \inf_{\pi} \sup_{\sigma} \mathcal{P}_v^{\sigma,\pi}(W) \tag{1.1}$$

Here σ and π ranges over the set of all strategies for player $\Box$ and player $\diamond$, respectively. However, the players do not necessarily have *optimal* strategies that would guarantee the outcome $val(v)$ or better against every strategy of the opponent. On the other hand, it follows directly from the above equation that each player has an ε-optimal strategy (see Definition 2.3) for every $\varepsilon > 0$. This means that if $\varrho \neq val(v)$, then one of the two players has a $(\triangleright\varrho)$-winning strategy for the game initiated in v. The situation when $\varrho = val(v)$ is more problematic, and to the best of our knowledge, the literature does not yet offer a general answer. Let us also note that for *finite-state* stochastic games and the "usual" classes of quantitative/qualitative Borel objectives (such as Büchi, Rabin, Street, etc.), the determinacy follows from the existence of optimal strategies (hence, the sup and inf in Equation 1.1 can be safely replaced with max and min, respectively). For classes of infinite-state stochastic games (such as stochastic BPA games considered in this paper), optimal strategies do not necessarily exist and the associated determinacy results must be proven by other methods.

Algorithmic issues for stochastic games with quantitative/qualitative winning objectives have been studied mainly for finite-state stochastic games. A lot of attention has been devoted to quantitative *reachability objectives*, even in the special case when $\varrho = \frac{1}{2}$. The problem whether player $\Box$ has a $(>\frac{1}{2})$-winning strategy is known to be in **NP** $\cap$ **co-NP**, but its membership to **P** is one of the long-standing open problems in algorithmic game theory [9, 20]. Later, more complicated qualitative/quantitative ω-regular winning objectives (such as Büchi, co-Büchi, Rabin, Street, Muller, etc.) were considered, and the complexity of the corresponding decision problems was analysed. We refer to [10, 6, 8, 7, 21, 19] for more details. As for infinite-state stochastic games, the attention has so far been focused on stochastic games induced by lossy channel systems [1, 2] and by pushdown automata (or, equivalently, recursive state machines) [14, 15, 13, 12, 4]. In the next paragraphs, we discuss the latter model in greater detail because these results are closely related to the results presented in this paper.

A *pushdown automaton (PDA)* (see, e.g., [16]) is equipped with a finite control unit and an unbounded stack. The dynamics is specified by a finite set of rules of the form $pX \hookrightarrow q\alpha$, where p, q are control states, X is a stack symbol, and α is a (possibly empty) sequence of stack symbols. A rule of the form $pX \hookrightarrow q\alpha$ is applicable to every configuration of the form $pX\beta$ and produces the configuration $q\alpha\beta$. If there are several rules with the same left-hand side, one of them must be

chosen, and the choice is appointed to player $\Box$, player $\Diamond$, or it is randomized. Technically, the set of all left-hand sides (i.e., pairs of the form pX) is split into three disjoint subsets $H_\Box$, $H_\Diamond$, and $H_\bigcirc$, and for all $pX \in H_\bigcirc$ there is a fixed probability distribution over the set of all rules of the form $pX \hookrightarrow q\alpha$. Thus, each PDA induces the associated infinite-state stochastic game where the vertices are PDA configurations and the edges are determined in the natural way. An important subclass of PDA is obtained by restricting the number of control states to 1. Such PDA are also known as *stateless* PDA or (mainly in concurrency theory) as BPA. PDA and BPA correspond to *recursive state machines (RSM)* and *1-exit RSM* respectively, in the sense that their descriptive powers are equivalent, and there are effective linear-time translations between the corresponding models.

In [13], the quantitative and qualitative *termination objective* for PDA and BPA stochastic games is examined (a terminating run is a run which hits a configuration with the empty stack; hence, termination is a special form of reachability). For BPA, it is shown that the vector of optimal values $(val(X), X \in \Gamma)$, where Γ is the stack alphabet, forms the least solution of an effectively constructible system of min-max equations. Moreover, both players have optimal MD strategies which depend only on the top-of-the-stack symbol of a given configuration (such strategies are called SMD, meaning Stackless MD). Hence, stochastic BPA games with quantitative/qualitative termination objectives are determined. Since the least solution of the constructed equational system can be encoded in first order theory of the reals, the existence of a $(\triangleright\varrho)$-winning strategy for player $\Box$ and player $\Diamond$ can be decided in polynomial space. In the same paper [13], the $\Sigma_2^P \cap \Pi_2^P$ upper complexity bound for the subclass of qualitative termination objectives is established. As for PDA games, it is shown that for every fixed $\varepsilon > 0$, the problem to distinguish whether the optimal value $val(pX)$ is equal to 1 or less than ε, is undecidable. The $\Sigma_2^P \cap \Pi_2^P$ upper bound for stochastic BPA games with qualitative termination objectives was improved to **NP** $\cap$ **co-NP** in [15]. In the same paper, it is also shown that the quantitative reachability problem for finite-state stochastic games (see above) is efficiently reducible to the qualitative termination problem for stochastic BPA games. Hence, the **NP** $\cap$ **co-NP** upper bound cannot be improved without a major breakthrough in algorithmic game theory. In the special case of stochastic BPA games where $H_\Diamond = \emptyset$ or $H_\Box = \emptyset$, the qualitative termination problem is shown to be in **P** (observe that if $H_\Diamond = \emptyset$ or $H_\Box = \emptyset$, then a given BPA induces an infinite-state Markov decision process and the goal of the only player is to maximize or minimize the termination probability, respectively). The results for Markov decision processes induced by BPA are generalized to (arbitrary) qualitative *reachability objectives* in [5], retaining the **P** upper complexity bound. In the same paper, it is also noted that the properties of reachability objectives are quite different from the ones of termination (in particular, there is no apparent way how to express the vector of optimal values as a solution of some recursive equational system, and the SMD determinacy result (see above) does not hold).

Our contribution: In this paper, we continue the study initiated in [14, 15, 13, 12, 4] and solve the qualitative reachability problem for unrestricted stochastic BPA games. Thus, we obtain a substantial generalization of the previous results.

We start by resolving the determinacy issue in Section 3, and this part of our work actually applies to arbitrary *finitely branching* stochastic games, where each vertex has only finitely many successors (BPA stochastic games are finitely branching). We show that finitely branching stochastic games with quantitative/qualitative reachability objectives are determined, i.e., in every vertex, one of the two players has a $(\triangleright\varrho)$-wining strategy. This is a consequence of several observations that are specific for reachability objectives and perhaps interesting on their own.

The main results of our paper, presented in Section 4, concern stochastic BPA games with qualitative reachability objectives. In the context of BPA, a reachability objective is specified by a *regular* set T of target configurations. We show that the problem of determining the winner in

stochastic BPA games with qualitative reachability objectives is in **NP** ∩ **co-NP**. Here we rely on the previously discussed results about qualitative termination [15] and use the corresponding algorithms as "black-box procedures" at appropriate places. We also rely on observations presented in [5] which were used to solve the simpler case with only one player. However, the full (two-player) case brings completely new complications that need to be tackled by new methods and ideas. Many "natural" hypotheses turned out to be incorrect (some of the interesting cases are documented by examples in Section 4). We also show that the sets of all configurations where player □ and player ◇ have a $(\triangleright\varrho)$-winning strategy (where $\varrho \in \{0, 1\}$) is effectively regular and the corresponding finite-state automata are effectively constructible by a deterministic polynomial-time algorithm with **NP** ∩ **co-NP** oracle. Finally, we also give an algorithm which *computes* a $(\triangleright\varrho)$-winning strategy if it exists. These strategies are randomized and memoryless, and they are also *effectively regular* in the sense that their functionality can effectively be encoded by finite-state automata (see Definition 4.3). Hence, winning strategies in stochastic BPA games with qualitative reachability objectives can be effectively implemented.

Due to space constraints, most of the proofs had to be omitted and can be found in the full version of this paper [3]. In the main body of the paper, we try to sketch the key ideas and provide some intuition behind the presented technical constructions.

2. Basic Definitions

In this paper, the set of all positive integers, non-negative integers, rational numbers, real numbers, and non-negative real numbers are denoted $\mathbb{N}$, $\mathbb{N}_0$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{R}^{\geq 0}$, respectively. For every finite or countably infinite set S, the symbol S^* denotes the set of all finite words over S. The length of a given word u is denoted $|u|$, and the individual letters in u are denoted $u(0), \cdots, u(|u| - 1)$. The empty word is denoted ε, where $|\varepsilon| = 0$. We also use S^+ to denote the set $S^* \setminus \{\varepsilon\}$. For every finite or countably infinite set M, a binary relation $\rightarrow \subseteq M \times M$ is *total* if for every $m \in M$ there is some $n \in M$ such that $m \rightarrow n$. A *path* in $\mathcal{M} = (M, \rightarrow)$ is a finite or infinite sequence $w = m_0, m_1, \ldots$ such that $m_i \rightarrow m_{i+1}$ for every i. The *length* of a finite path $w = m_0, \ldots, m_i$, denoted *length*(w), is $i + 1$. We also use $w(i)$ to denote the element m_i of w, and w_i to denote the path $m_i, m_{i+1}, \ldots$ (by writing $w(i) = m$ or w_i we implicitly impose the condition that *length*$(w) \geq i+1$). A given $n \in M$ is *reachable* from a given $m \in M$, written $m \rightarrow^* n$, if there is a finite path from m to n. A *run* is an infinite path. The sets of all finite paths and all runs in $\mathcal{M}$ are denoted *FPath*$(\mathcal{M})$ and *Run*$(\mathcal{M})$, respectively. Similarly, the sets of all finite paths and runs that start in a given $m \in M$ are denoted *FPath*$(\mathcal{M}, m)$ and *Run*$(\mathcal{M}, m)$, respectively.

Now we recall basic notions of probability theory. Let A be a finite or countably infinite set. A *probability distribution* on A is a function $f : A \rightarrow \mathbb{R}^{\geq 0}$ such that $\sum_{a \in A} f(a) = 1$. A distribution f is *rational* if $f(a) \in \mathbb{Q}$ for every $a \in A$, *positive* if $f(a) > 0$ for every $a \in A$, and *Dirac* if $f(a) = 1$ for some $a \in A$. The set of all distributions on A is denoted $\mathcal{D}(A)$.

A *σ-field* over a set X is a set $\mathcal{F} \subseteq 2^X$ that includes X and is closed under complement and countable union. A *measurable space* is a pair $(X, \mathcal{F})$ where X is a set called *sample space* and $\mathcal{F}$ is a σ-field over X. A *probability measure* over a measurable space $(X, \mathcal{F})$ is a function $\mathcal{P} : \mathcal{F} \rightarrow \mathbb{R}^{\geq 0}$ such that, for each countable collection $\{X_i\}_{i \in I}$ of pairwise disjoint elements of $\mathcal{F}$, $\mathcal{P}(\bigcup_{i \in I} X_i) = \sum_{i \in I} \mathcal{P}(X_i)$, and moreover $\mathcal{P}(X) = 1$. A *probability space* is a triple $(X, \mathcal{F}, \mathcal{P})$ where $(X, \mathcal{F})$ is a measurable space and $\mathcal{P}$ is a probability measure over $(X, \mathcal{F})$.

Definition 2.1. A *Markov chain* is a triple $\mathcal{M} = (M, \longrightarrow, Prob)$ where M is a finite or countably infinite set of *states*, $\longrightarrow \subseteq M \times M$ is a total *transition relation*, and *Prob* is a function which to each $s \in M$ assigns a positive probability distribution over the set of its outgoing transitions.

In the rest of this paper, we write $s \xrightarrow{x} t$ whenever $s \longrightarrow t$ and $Prob((s,t)) = x$. Each $w \in FPath(\mathcal{M})$ determines a *basic cylinder* $Run(\mathcal{M}, w)$ which consists of all runs that start with w. To every $s \in M$ we associate the probability space $(Run(\mathcal{M}, s), \mathcal{F}, \mathcal{P})$ where $\mathcal{F}$ is the σ-field generated by all basic cylinders $Run(\mathcal{M}, w)$ where w starts with s, and $\mathcal{P} : \mathcal{F} \to \mathbb{R}^{\geq 0}$ is the unique probability measure such that $\mathcal{P}(Run(\mathcal{M}, w)) = \Pi_{i=0}^{m-1} x_i$ where $w = s_0, \cdots, s_m$ and $s_i \xrightarrow{x_i} s_{i+1}$ for every $0 \leq i < m$ (if $m = 0$, we put $\mathcal{P}(Run(\mathcal{M}, w)) = 1$).

Definition 2.2. A *stochastic game* is a tuple $G = (V, \mapsto, (V_\Box, V_\Diamond, V_\bigcirc), Prob)$ where V is a finite or countably infinite set of *vertices*, $\mapsto \subseteq V \times V$ is a total *edge relation*, $(V_\Box, V_\Diamond, V_\bigcirc)$ is a partition of V, and *Prob* is a *probability assignment* which to each $v \in V_\bigcirc$ assigns a positive probability distribution on the set of its outgoing transitions. We say that G is *finitely branching* if for each $v \in V$ there are only finitely many $u \in V$ such that $v \mapsto u$.

A stochastic game is played by two players, $\Box$ and $\Diamond$, who select the moves in the vertices of $V_\Box$ and $V_\Diamond$, respectively. Let $\odot \in \{\Box, \Diamond\}$. A *strategy* for player $\odot$ is a function which to each $wv \in V^* V_\odot$ assigns a probability distribution on the set of outgoing edges of v. The set of all strategies for player $\Box$ and player $\Diamond$ is denoted Σ and Π, respectively. We say that a strategy τ is *memoryless (M)* if $\tau(wv)$ depends just on the last vertex v, and *deterministic (D)* if $\tau(wv)$ is a Dirac distribution for all wv. Strategies that are not necessarily memoryless are called *history-dependent (H)*, and strategies that are not necessarily deterministic are called *randomized (R)*. Hence, we can define the following four classes of strategies: MD, MR, HD, and HR, where MD $\subseteq$ HD $\subseteq$ HR and MD $\subseteq$ MR $\subseteq$ HR, but MR and HD are incomparable.

Each pair of strategies $(\sigma, \pi) \in \Sigma \times \Pi$ determines a unique *play* of the game G, which is a Markov chain $G(\sigma, \pi)$ where V^+ is the set of states, and $wu \xrightarrow{x} wuu'$ iff $u \mapsto u'$ and one of the following conditions holds:

- $u \in V_\Box$ and $\sigma(wu)$ assigns x to $u \mapsto u'$, where $x > 0$;
- $u \in V_\Diamond$ and $\pi(wu)$ assigns x to $u \mapsto u'$, where $x > 0$;
- $u \in V_\bigcirc$ and $u \xrightarrow{x} u'$.

Let $T \subseteq V$ be a set of *target* vertices. For each pair of strategies $(\sigma, \pi) \in \Sigma \times \Pi$ and every $v \in V$, let $\mathcal{P}_v^{\sigma,\pi}(Reach(T))$ be the probability of all $w \in Run(G(\sigma, \pi), v)$ such that w visits some $u \in T$ (technically, this means that $w(i) \in V^* T$ for some $i \in \mathbb{N}_0$). We say that a given $v \in V$ *has a value* if $\sup_{\sigma \in \Sigma} \inf_{\pi \in \Pi} \mathcal{P}_v^{\sigma,\pi}(Reach(T)) = \inf_{\pi \in \Pi} \sup_{\sigma \in \Sigma} \mathcal{P}_v^{\sigma,\pi}(Reach(T))$. If v has a value, then $val(v)$ denotes the *value of* v defined by this equality. Since the set of all runs that visit a vertex of T is obviously Borel, we can apply the powerful result of Martin [18] (see also Theorem 3.3) and conclude that *every* $v \in V$ has a value.

Definition 2.3. Let $\varepsilon \geq 0$. We say that

- $\sigma \in \Sigma$ is *ε-optimal* (or *ε-optimal maximizing*) if $\mathcal{P}_v^{\sigma,\pi}(Reach(T)) \geq val(v) - \varepsilon$ for all $\pi \in \Pi$;
- $\pi \in \Pi$ is *ε-optimal* (or *ε-optimal minimizing*) if $\mathcal{P}_v^{\sigma,\pi}(Reach(T)) \leq val(v) + \varepsilon$ for all $\sigma \in \Sigma$.

A 0-optimal strategy is called *optimal*. A *(quantitative) reachability objective* is a pair $(T, \triangleright\varrho)$ where $T \subseteq V$ and $\triangleright\varrho$ is a probability constraint, i.e., $\triangleright \in \{>, \geq\}$ and $\varrho \in [0, 1]$. If $\varrho \in \{0, 1\}$, then the objective is *qualitative*. We say that

- $\sigma \in \Sigma$ is *$(\triangleright\varrho)$-winning* if $\mathcal{P}_v^{\sigma,\pi}(Reach(T)) \triangleright \varrho$ for all $\pi \in \Pi$;
- $\pi \in \Pi$ is *$(\triangleright\varrho)$-winning* if $\mathcal{P}_v^{\sigma,\pi}(Reach(T)) \not\triangleright \varrho$ for all $\sigma \in \Sigma$.

3. The Determinacy of Stochastic Games with Reachability Objectives

In this section we show that finitely-branching stochastic games with quantitative/qualitative reachability objectives are *determined* in the sense that for every quantitative reachability objective $(T, \triangleright\varrho)$ and every vertex v of a finitely branching stochastic game, one of the two players has a $(\triangleright\varrho)$-winning strategy.

For the rest of this section, let us fix a finitely branching game $G = (V, \mapsto, (V_\Box, V_\Diamond, V_\bigcirc), \mathit{Prob})$ and a set of target vertices T. Also, for every $n \in \mathbb{N}_0$ and a pair of strategies $(\sigma, \pi) \in \Sigma \times \Pi$, let $\mathcal{P}_v^{\sigma,\pi}(\mathit{Reach}_n(T))$ be the probability of all runs $w \in \mathit{Run}(G(\sigma,\pi), v)$ such that w visits some $u \in T$ in at most n transitions (clearly, $\mathcal{P}_v^{\sigma,\pi}(\mathit{Reach}(T)) = \lim_{n\to\infty} \mathcal{P}_v^{\sigma,\pi}(\mathit{Reach}_n(T))$).

To keep this paper self-contained, we start by giving a simple proof of Martin's weak determinacy result (Equation 1.1) for the special case of finitely-branching games with reachability objectives. For every $v \in V$ and $i \in \mathbb{N}_0$, we define $\mathcal{V}_i(v) \in \mathbb{N}_0$ inductively as follows: $\mathcal{V}_0(v)$ is equal either to 1 or 0, depending on whether $v \in T$ or not, respectively. $\mathcal{V}_{i+1}(v)$ (for $v \notin T$) is equal either to $\max\{\mathcal{V}_i(u) \mid v \mapsto u\}$, $\min\{\mathcal{V}_i(u) \mid v \mapsto u\}$, or $\sum_{v \xrightarrow{x} u} x \cdot \mathcal{V}_i(u)$, depending on whether $v \in V_\Box$, $v \in V_\Diamond$, or $v \in V_\bigcirc$, respectively. (For $v \in T$ we put $\mathcal{V}_{i+1}(v) = 1$.) Further, put $\mathcal{V}(v) = \lim_{i\to\infty} \mathcal{V}_i(v)$ (note that the limit exists because the sequence $\mathcal{V}_0(v), \mathcal{V}_1(v), \ldots$ is non-decreasing and bounded). A straightforward induction on i reveals that

$$\mathcal{V}_i(v) \quad = \quad \max_{\sigma\in\Sigma} \min_{\pi\in\Pi} \mathcal{P}_v^{\sigma,\pi}(\mathit{Reach}_i(T)) \quad = \quad \min_{\pi\in\Pi} \max_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi}(\mathit{Reach}_i(T))$$

Also observe that, for every $i \in \mathbb{N}_0$, there are fixed HD strategies $\sigma_i \in \Sigma$ and $\pi_i \in \Pi$ such that for every $\pi \in \Pi$ and $\sigma \in \Sigma$ we have that $\mathcal{P}_v^{\sigma,\pi_i}(\mathit{Reach}_i(T)) \le \mathcal{V}_i(v) \le \mathcal{P}_v^{\sigma_i,\pi}(\mathit{Reach}_i(T))$.

Theorem 3.1. *Every $v \in V$ has a value and $\mathit{val}(v) = \mathcal{V}(v)$.*

Proof. One can easily verify that

$$\mathcal{V}(v) \quad \le \quad \sup_{\sigma\in\Sigma} \inf_{\pi\in\Pi} \mathcal{P}_v^{\sigma,\pi}(\mathit{Reach}(T)) \quad \le \quad \inf_{\pi\in\Pi} \sup_{\sigma\in\Sigma} \mathcal{P}_v^{\sigma,\pi}(\mathit{Reach}(T)) \tag{3.1}$$

Hence, it suffices to show that, for every $v \in V$, player $\Diamond$ has a $(>\mathcal{V}(v))$-winning HD strategy $\bar\pi$ in v.

For every $i \in \mathbb{N}$, let W_i be the set of all $w \in V^*V_\Diamond$ such that $w(0) = v$, $\mathit{length}(w) = i$, and $w(i) \mapsto w(i{+}1)$ for every $0 \le i < \mathit{length}(w)$. The strategy $\bar\pi$ is defined inductively, together with an auxiliary set $\Pi_i \subseteq \Pi$. We start by putting $\Pi_1 = \{\pi_i \mid i \in \mathbb{N}_0\}$. Now assume that Π_i has already been defined. For every $wu \in W_i$, let us fix an edge $u \mapsto u'$ such that $\pi(wu)(u \mapsto u') = 1$ for infinitely many $\pi \in \Pi_i$ (observe that there must be such an edge because G is finitely branching). We put $\bar\pi(wu)(u \mapsto u') = 1$ and $\Pi_{i+1} = \{\pi \in \Pi_i \mid \pi(wu)(u \mapsto u') = 1\}$.

We claim that for every $\sigma \in \Sigma$ we have that $\mathcal{P}_v^{\sigma,\bar\pi}(\mathit{Reach}(T)) \le \mathcal{V}(v)$. Assume the opposite. Then there is $\bar\sigma \in \Sigma$ such that $\mathcal{P}_v^{\bar\sigma,\bar\pi}(\mathit{Reach}(T)) = \varrho > \mathcal{V}(v)$. Further, there is some $k \in \mathbb{N}$ such that $\mathcal{P}_v^{\bar\sigma,\bar\pi}(\mathit{Reach}_k(T)) > \mathcal{V}(v) + (\varrho - \mathcal{V}(v))/2$. It follows directly from the definition of $\bar\pi$ that there is some $m \in \mathbb{N}, m > k$ such that $\pi_m \in \Pi_m$ and $\bar\pi(w) = \pi_m(w)$ for every $w \in W_m$. Hence, $\mathcal{P}_v^{\bar\sigma,\pi_m}(\mathit{Reach}_m(T)) > \mathcal{V}(v) + (\varrho - \mathcal{V}(v))/2 > \mathcal{V}(v)$, which contradicts the definition of $\mathcal{V}$. ∎

The characterization of $\mathit{val}(v)$ as a limit of $\mathcal{V}_i(v)$ has the following important consequence:

Lemma 3.2. *For every fixed vertex $v \in V$, we have that*

$$\forall\varepsilon{>}0 \;\; \exists\sigma \in \Sigma \;\; \exists n \in \mathbb{N} \;\; \forall\pi \in \Pi \;:\; \mathcal{P}_v^{\sigma,\pi}(\mathit{Reach}_n(T)) > \mathit{val}(v) - \varepsilon$$

Proof. It suffices to choose a sufficiently large $n \in \mathbb{N}$ and put $\sigma = \sigma_n$. ∎

Note that from the proof of Theorem 3.1 we obtain a HD strategy $\bar{\pi} \in \Pi$ such that $\forall v \in V$ and $\forall \sigma \in \Sigma$ we have that $\mathcal{P}_v^{\sigma,\bar{\pi}}(Reach(T)) \leq val(v)$. This result can be strengthened to MD strategies.

Theorem 3.3. *There is a MD strategy $\pi \in \Pi$ such that for every $v \in V$ and every $\sigma \in \Sigma$ we have that $\mathcal{P}_v^{\sigma,\pi}(Reach(T)) \leq val(v)$. That is, π is an optimal minimizing strategy in every vertex.*

Theorem 3.4 (Determinacy)**.** *Let $v \in V$ and let $(T, \triangleright\varrho)$ be a (quantitative) reachability objective. Then one of the two players has a $(\triangleright\varrho)$-winning strategy in v.*

Proof outline. We prove that if player $\diamond$ does not have a $\triangleright\varrho$-winning strategy, then player $\Box$ has a $\triangleright\varrho$-winning strategy. That is, we prove the implication

$$\forall \pi \in \Pi \ \exists \sigma \in \Sigma \ : \ \mathcal{P}_v^{\sigma,\pi}(Reach(T)) \triangleright \varrho \quad \Rightarrow \quad \exists \sigma \in \Sigma \ \forall \pi \in \Pi \ : \ \mathcal{P}_v^{\sigma,\pi}(Reach(T)) \triangleright \varrho \qquad (3.2)$$

If $\triangleright$ is $>$ or $val(v) \neq \varrho$, then this follows easily by Theorem 3.3. For the constraint ≥ 0 the statement is trivial. Now suppose that $\triangleright$ is $\geq$ and $\varrho = val(v) > 0$, and assume that the left-hand side in (3.2) holds. Observe that we can safely restrict the set of edges available to player $\Box$ to those $u \mapsto u'$ where $val(u') = val(u)$. Using the left-hand side of (3.2), one can show that for every $s \in V$, the value $val(s)$ stays unchanged in the new game obtained by applying this restriction. Due to Lemma 3.2, to every $s \in V$ in the new game we can associate a strategy $\sigma_s \in \Sigma$ and $n_s \in \mathbb{N}$ such that for every $\pi \in \Pi$ we have that $\mathcal{P}_s^{\sigma_s,\pi}(Reach_{n_s}(T)) > val(s)/2$. The $\geq\varrho$-winning strategy σ for player $\Box$ is obtained by "iterating" the strategies σ_s in the following sense: we start with σ_v, and after performing a path w of length n_v, we change the strategy to σ_s where s is the last vertex visited by w. The strategy σ_s is used for the next n_s transition, and then we perform another "iteration". Observe that each round of this "iteration" decreases the probability that T is *not* reached by a factor of $1/2$, independently of the strategy of player $\diamond$. ∎

4. Qualitative Reachability in Stochastic BPA Games

Stochastic BPA games correspond to stochastic games induced by stateless pushdown automata or 1-exit recursive state machines (see Section 1). A formal definition follows.

Definition 4.1. A *stochastic BPA* game is a tuple $\Delta = (\Gamma, \hookrightarrow, (\Gamma_\Box, \Gamma_\diamond, \Gamma_\bigcirc), Prob)$ where Γ is a finite *stack alphabet*, $\hookrightarrow \subseteq \Gamma \times \Gamma^{\leq 2}$ is a finite set of *rules* (where $\Gamma^{\leq 2} = \{w \in \Gamma^* : |w| \leq 2\}$) such that for each $X \in \Gamma$ there is some rule $X \hookrightarrow \alpha$, $(\Gamma_\Box, \Gamma_\diamond, \Gamma_\bigcirc)$ is a partition of Γ, and *Prob* is a *probability assignment* which to each $X \in \Gamma_\bigcirc$ assigns a rational positive probability distribution on the set of all rules of the form $X \hookrightarrow \alpha$.

A *configuration* of Δ is a word $\alpha \in \Gamma^*$, which can intuitively be interpreted as the current stack content where the leftmost symbol of α is on top of the stack. Each stochastic BPA game $\Delta = (\Gamma, \hookrightarrow, (\Gamma_\Box, \Gamma_\diamond, \Gamma_\bigcirc), Prob)$ determines a unique stochastic game $G_\Delta = (\Gamma^*, \mapsto, (\Gamma_\Box\Gamma^*, \Gamma_\diamond\Gamma^*, \Gamma_\bigcirc\Gamma^* \cup \{\varepsilon\}), Prob_\Delta)$ where the transitions of $\mapsto$ are determined as follows: $\varepsilon \mapsto \varepsilon$, and $X\beta \mapsto \alpha\beta$ iff $X \hookrightarrow \alpha$. The probability assignment $Prob_\Delta$ is the natural extension of *Prob*, i.e., $\varepsilon \overset{1}{\mapsto} \varepsilon$ and for all $X \in \Gamma_\bigcirc$ we have that $X\beta \overset{x}{\mapsto} \alpha\beta$ iff $X \overset{x}{\hookrightarrow} \alpha$.

In this section we consider stochastic BPA games with qualitative termination objectives $(T, \triangleright\varrho)$ where $T \subseteq \Gamma^*$ is a *regular* set of configurations. For technical convenience, we define the size of T as the size of the minimal deterministic finite-state automaton $\mathscr{A}_T = (Q, q_0, \delta, F)$ which recognizes the *reverse* of T (if we view configurations as stacks, this corresponds to bottom-up direction). Note that the automaton $\mathscr{A}_T$ can be simulated on-the-fly in Δ by employing standard techniques (see, e.g., [11]). That is, the stack alphabet is extended to $\Gamma \times Q$ and the rules are adjusted accordingly (for

example, if $X \hookrightarrow YZ$, then for every $q \in Q$ the extended BPA game has a rule $(X, q) \hookrightarrow (Y, r)(Z, q)$ where $\delta(q, Z) = r$). Note that the on-the-fly simulation of $\mathscr{A}_T$ in Δ does not affect the way how the game is played, and the size of the extended game in polynomial in $|\Delta|$ and $|\mathscr{A}_T|$. The main advantage of this simulation is that the information whether a current configuration belongs to T or not can now be deduced just by looking at the symbol on top of the stack. This leads to an important technical simplification in the definition of T:

Definition 4.2. We say that $T \subseteq \Gamma^*$ is *simple* if $\varepsilon \notin T$ and there is $\Gamma_T \subseteq \Gamma$ such that for every $X\alpha \in \Gamma^+$ we have that $X\alpha \in T$ iff $X \in \Gamma_T$.

Note that the requirement $\varepsilon \notin T$ in the previous definition is not truly restrictive, because each BPA can be equipped with a fresh bottom-of-the-stack symbol which cannot be removed. Hence, we can safely restrict ourselves just to simple sets of target configurations. All of the obtained results (including the complexity bounds) are valid also for regular sets of target configurations.

Since stochastic BPA games have infinitely many vertices, even memoryless strategies are not necessarily finitely representable. It turns out that the winning strategies for both players in stochastic BPA games with qualitative reachability objectives are (effectively) *regular* in the following sense:

Definition 4.3. Let $\Delta = (\Gamma, \hookrightarrow, (\Gamma_\Box, \Gamma_\Diamond, \Gamma_\bigcirc), \mathit{Prob})$ be a stochastic BPA game, and let $\odot \in \{\Box, \Diamond\}$. We say that a strategy τ for player $\odot$ is *regular* if there is a deterministic finite-state automaton $\mathscr{A}$ over the alphabet Γ such that, for every $X\alpha \in \Gamma_\odot\Gamma^*$, the value of $\tau(X\alpha)$ depends just on the control state entered by $\mathscr{A}$ after reading the reverse of $X\alpha$ (i.e., the automaton $\mathscr{A}$ reads the stack bottom-up).

For the rest of this section, we fix a stochastic BPA game $\Delta = (\Gamma, \hookrightarrow, (\Gamma_\Box, \Gamma_\Diamond, \Gamma_\bigcirc), \mathit{Prob})$ and a simple set T of target configurations. Since we are interested just in reachability objectives, we can safely assume that for every $R \in \Gamma_T$, the only rule where R appears on the left-hand side is $R \hookrightarrow R$ (this assumption simplifies the formulation of some claims). We use T_ε to denote the set $T \cup \{\varepsilon\}$, and we also slightly abuse the notation by writing ε instead of $\{\varepsilon\}$ at some places (particularly in expressions such as $\mathit{Reach}(\varepsilon)$).

For a given set $C \subseteq \Gamma^*$ and a given qualitative probability constraint $\triangleright\varrho$, we use $[C]^{\triangleright\varrho}_\Box$ and $[C]^{\triangleright\varrho}_\Diamond$ to denote the set of all $\alpha \in \Gamma^*$ from which player $\Box$ and player $\Diamond$ has a $(\triangleright\varrho)$-winning strategy in the game Δ with the reachability objective $(C, \triangleright\varrho)$, respectively. Observe that $[C]^{\triangleright\varrho}_\Box = \Gamma^* \setminus [C]^{\triangleright\varrho}_\Diamond$ due to the determinacy results presented in Section 3.

In the forthcoming subsections we examine the sets $[T]^{\triangleright\varrho}_\Box$ for the two meaningful qualitative probability constraints >0 and $=1$ (observe that $[T]^{\geq 0}_\Box = \Gamma^*$ and $[T]^{>1}_\Box = \emptyset$). We show that the membership to $[T]^{>0}_\Box$ and $[T]^{=1}_\Box$ is in **P** and **NP** $\cap$ **co-NP**, respectively. The same holds for the sets $[T]^{>0}_\Diamond$ and $[T]^{=1}_\Diamond$, respectively. Further, we show that all of these sets are effectively regular, and that $(\triangleright\varrho)$-winning strategies for both players are effectively computable. The associated upper complexity bounds are essentially the same as above.

4.1. The Set $[T]^{>0}_\Box$

We start by observing that the sets $[T]^{>0}_\Box$ and $[T]^{>0}_\Diamond$ are regular, and the associated finite-state automata have a fixed number of control states. A proof of this observation is actually straightforward.

Proposition 4.4. *Let* $\mathscr{A} = [T]^{>0}_\Box \cap \Gamma$ *and* $\mathscr{B} = [T_\varepsilon]^{>0}_\Box \cap \Gamma$. *Then* $[T]^{>0}_\Box = \mathscr{B}^*\mathscr{A}\Gamma^*$ *and* $[T_\varepsilon]^{>0}_\Box = \mathscr{B}^*\mathscr{A}\Gamma^* \cup \mathscr{B}^*$. *Consequently,* $[T]^{>0}_\Diamond = \Gamma^* \setminus [T]^{>0}_\Box = (\mathscr{B} \setminus \mathscr{A})^* \cup (\mathscr{B} \setminus \mathscr{A})^*(\Gamma \setminus \mathscr{B})\Gamma^*$.

Our next proposition says how to compute the sets $\mathscr{A}$ and $\mathscr{B}$.

Proposition 4.5. *The pair $(\mathscr{A},\mathscr{B})$ is the least fixed-point of the function $F : (2^\Gamma \times 2^\Gamma) \to (2^\Gamma \times 2^\Gamma)$, where $F(A,B) = (\hat{A},\hat{B})$ is defined as follows:*

$$\begin{aligned}
\hat{A} &= \Gamma_T \cup A \cup \{X \in \Gamma_\Box \cup \Gamma_\bigcirc \mid \textit{ there is } X \hookrightarrow \beta \textit{ such that } \beta \in B^*A\Gamma^*\} \\
&\cup \{X \in \Gamma_\Diamond \mid \textit{ for all } X \hookrightarrow \beta \textit{ we have that } \beta \in B^*A\Gamma^*\} \\
\hat{B} &= \Gamma_T \cup B \cup \{X \in \Gamma_\Box \cup \Gamma_\bigcirc \mid \textit{ there is } X \hookrightarrow \beta \textit{ such that } \beta \in B^*A\Gamma^* \cup B^*\} \\
&\cup \{X \in \Gamma_\Diamond \mid \textit{ for all } X \hookrightarrow \beta \textit{ we have that } \beta \in B^*A\Gamma^* \cup B^*\}
\end{aligned}$$

Since the least fixed-point of the function F defined in Proposition 4.5 is computable in polynomial time, the finite-state automata recognizing the sets $[T]^{>0}_\Box$ and $[T]^{>0}_\Diamond$ are computable in polynomial time. Thus, we obtain the following theorem:

Theorem 4.6. *The membership to $[T]^{>0}_\Box$ and $[T]^{>0}_\Diamond$ is decidable in polynomial time. Both sets are effectively regular, and the associated finite-state automata are constructible in polynomial time. Further, there are regular strategies $\sigma \in \Sigma$ and $\pi \in \Pi$ constructible in polynomial time that are (>0)-winning in every configuration of $[T]^{>0}_\Box$ and $[T]^{>0}_\Diamond$, respectively.*

4.2. The Set $[T]^{=1}_\Box$

The results presented in this subsection constitute the very core of this paper. The problems are more complicated than in the case of $[T]^{>0}_\Box$, and several deep observations are needed to tackle them. We start by showing that the sets $[T]^{=1}_\Box$ and $[T]^{=1}_\Diamond$ are regular.

Proposition 4.7. *Let $\mathscr{A} = [T_\varepsilon]^{=1}_\Diamond \cap \Gamma$, $\mathscr{B} = [T_\varepsilon]^{=1}_\Box \cap [T]^{=1}_\Diamond \cap \Gamma$, $\mathscr{C} = [T]^{=1}_\Box \cap \Gamma$. Then $[T]^{=1}_\Box = \mathscr{B}^*\mathscr{C}\Gamma^*$ and $[T]^{=1}_\Diamond = \mathscr{B}^*\mathscr{A}\Gamma^* \cup \mathscr{B}^*$.*

Proposition 4.7 can be proven by a straightforward induction on the length of configurations. Observe that if there is an algorithm which computes the set $\mathscr{A} = [T_\varepsilon]^{=1}_\Diamond \cap \Gamma$ for an arbitrary stochastic BPA game, then this algorithm can also be used to compute the set $[T]^{=1}_\Diamond \cap \Gamma$ (this is because $X \in [T]^{=1}_\Diamond$ iff $\hat{X} \in [\hat{T}_\varepsilon]^{=1}_\Diamond$, where $[\hat{T}_\varepsilon]^{=1}_\Diamond$ is considered in a stochastic BPA game $\hat{\Delta}$ obtained from Δ by adding two fresh stochastic symbols $\hat{X}, Z$ together with the rules $\hat{X} \xhookrightarrow{1} XZ$, $Z \xhookrightarrow{1} Z$, and setting $\hat{T} = T$). Due to Theorem 3.4, we have that $\mathscr{C} = \Gamma \setminus ([T]^{=1}_\Diamond \cap \Gamma)$, and thus we can compute also the set $\mathscr{C}$. Since $\mathscr{B} = \Gamma \setminus (\mathscr{A} \cup \mathscr{C})$ (again by Theorem 3.4), we can also compute the set $\mathscr{B}$. Hence, the core of the problem is to design an algorithm which computes the set $\mathscr{A}$.

In the next definition we introduce the crucial notion of a *terminal* set of stack symbols, which plays a key role in our considerations.

Definition 4.8. A set $M \subseteq \Gamma$ is *terminal* if the following conditions are satisfied:

- $\Gamma_T \cap M = \emptyset$;
- for every $Z \in M \cap (\Gamma_\Box \cup \Gamma_\bigcirc)$ and every rule of the form $Z \hookrightarrow \alpha$ we have that $\alpha \in M^*$;
- for every $Z \in M \cap \Gamma_\Diamond$ there is a rule $Z \hookrightarrow \alpha$ such that $\alpha \in M^*$.

Since $\emptyset$ is terminal and the union of two terminal sets is terminal, there is the greatest terminal set that will be denoted C in the rest of this section. Also note that C determines a unique stochastic BPA game Δ_C obtained from Δ by restricting the set of stack symbols to C and including all rules $X \hookrightarrow \alpha$ where $X, \alpha \in C^*$. The set of rules of Δ_C is denoted $\hookrightarrow_C$. The probability of stochastic rules in Δ_C is the same as in Δ.

Definition 4.9. A stack symbol $Y \in \Gamma$ is a *witness* if one of the following conditions is satisfied:

(1) $Y \in [T_\varepsilon]^{>0}_\diamond$;

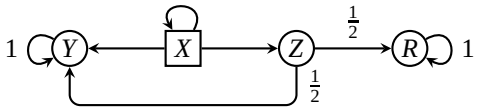
(2) $Y \in C$ and $Y \in [\varepsilon]^{=1}_\diamond$, where the set $[\varepsilon]^{=1}_\diamond$ is computed in Δ_C.

The set of all witnesses is denoted W.

Observe that the problem whether $Y \in W$ for a given $Y \in \Gamma$ is decidable in **NP**∩**co-NP**, because Condition (1) is decidable in **P** due to Theorem 4.6, the set C is computable in polynomial time, and the membership to $[\varepsilon]^{=1}_\diamond$ is in **NP** ∩ **co-NP** due to [15] (this is the only place where we use the decision algorithm for qualitative termination designed in [15]).

Obviously, $W \subseteq \mathscr{A}$. One may be tempted to think that the set $\mathscr{A}$ is just the *attractor* of W, denoted $Att(W)$, which consists of all $V \in \Gamma$ from which player $\diamond$ can enforce visiting a witness with a positive probability (i.e., $V \in Att(W)$ iff $\exists \pi \in \Pi \ \forall \sigma \in \Sigma : \mathcal{P}^{\sigma,\pi}_V(Reach(W\Gamma^*)) > 0$). However, this is not true, as it is demonstrated in the following example:

Example 4.10. Consider a stochastic BPA game $\hat{\Delta} = (\{X, Y, Z, R\}, \hookrightarrow, (\{X\}, \emptyset, \{Y, Z, R\}), Prob)$, where $X \hookrightarrow X$, $X \hookrightarrow Y$, $X \hookrightarrow Z$, $Y \overset{1}{\hookrightarrow} Y$, $Z \overset{1/2}{\hookrightarrow} Y$, $Z \overset{1/2}{\hookrightarrow} R$, $R \overset{1}{\hookrightarrow} R$, and the set T_Γ contains just R. The game is initiated in X, and the relevant part of $G_{\hat{\Delta}}$ (reachable from X) is shown in the following figure:



Observe that $\mathscr{A} = \{X, Y, Z\}$, $C = W = \{Y\}$, but $Att(\{Y\}) = \{Z, Y\}$.

In Example 4.10, the problem is that player $\square$ can use a strategy which always selects the rule $X \hookrightarrow X$ with probability one, and player $\diamond$ has no way to influence this. Nevertheless, observe that player $\square$ has essentially two options: he either enters a symbol of $Att(\{Y\})$, or he performs the loop $X \hookrightarrow X$ forever. The second possibility can be analyzed by "cutting off" the set $Att(\{Y\})$ and recomputing the set of all witnesses together with its attractor in the resulting stochastic BPA game, which contains only X and the rule $X \hookrightarrow X$. Observe that X is a witness in this game, and hence it can be safely added to the set $\mathscr{A}$. Thus, the computation of the set $\mathscr{A}$ for the stochastic BPA game $\hat{\Delta}$ is completed.

For general stochastic BPA games, the algorithm for computing the set $\mathscr{A}$ proceeds by initiating $\mathscr{A}$ to $\emptyset$ and then repeatedly computing the set $Att(W)$, setting $\mathscr{A} := \mathscr{A} \cup Att(W)$, and "cutting off" the set $Att(W)$ from the game. This goes on until the game or the set $Att(W)$ becomes empty. The way how $Att(W)$ is "cut off" from the current game is described below. First, let us present an important (and highly non-trivial) result which states the following:

Proposition 4.11. *If $\mathscr{A} \neq \emptyset$, then $W \neq \emptyset$.*

Proof outline. We show that if $W = \emptyset$, then there is a MR strategy $\sigma \in \Sigma$ such that for every $X \in \Gamma$ and every $\pi \in \Pi$ we have that $\mathcal{P}^{\sigma,\pi}_X(Reach(T_\varepsilon)) = 1$. In particular, this means that $\mathscr{A} = \emptyset$.

Since $W = \emptyset$, the condition of Definition 4.9 does not hold for any $Y \in \Gamma$, which in particular means that for all $Y \in C$ we have that $Y \notin [\varepsilon]^{=1}_\diamond$, i.e., $Y \in [\varepsilon]^{=1}_\square$ by Theorem 3.4 (here, the sets $[\varepsilon]^{=1}_\diamond$ and $[\varepsilon]^{=1}_\square$ are considered in the game Δ_C). Due to [13], there exists a SMD strategy σ_T for player $\square$ in Δ_C such that for every $Y \in C$ and every strategy π of player $\diamond$ in Δ_C we have that $\mathcal{P}^{\sigma_T,\pi}(Reach(\varepsilon)) = 1$. Now we define the promised MR strategy $\sigma \in \Sigma$ as follows: for a given $X\alpha \in \Gamma_\square\Gamma^*$, we put $\sigma(X\alpha) = \sigma_T(X\alpha)$ if $X\alpha$ starts with some $\beta \in C^*$ where $|\beta| > |\Delta|$. Otherwise, $\sigma(X\alpha)$ returns the uniform probability distribution over the outgoing transitions of $X\alpha$.

Now, let us fix some strategy $\pi \in \Pi$. Our goal is to show that $\mathcal{P}_X^{\sigma,\pi}(Reach(T_\varepsilon)) = 1$. By analyzing the play $G_\Delta(\sigma,\pi)$, one can show that there is a set of runs $V \subseteq Run(G_\Delta(\sigma,\pi), X)$ and a set of rules $\hookrightarrow_V \subseteq \hookrightarrow$ such that

(A) $\mathcal{P}(V) > 0$, $\hookrightarrow_V \subseteq \hookrightarrow_C$, and for every $w \in V$ we have that w does not visit T_ε and the set of rules that are used infinitely often in w is exactly $\hookrightarrow_V$.

Observe that each $w \in V$ has a finite prefix v_w such that the rules of $\hookrightarrow \setminus \hookrightarrow_C$ are used only in v_w. Further, we can partition the runs of V into countably many sets according to this prefix. One of these sets must have a positive probability, and hence we can conclude that there is $U \subseteq V$ and a finite path $v \in FPath(X)$ such that

(B) $\mathcal{P}(U) > 0$, and each $w \in U$ satisfies the following: w starts with v, the rules of $\hookrightarrow \setminus \hookrightarrow_C$ are used only in the prefix v of w, and the length of every configuration of w visited after the prefix v is at least as large as the length of the last configuration in the prefix v (the last condition still requires a justification which is omitted in here).

We show that $\mathcal{P}(U) = 0$, which is a contradiction. Roughly speaking, this is achieved by observing that, after performing the prefix v, the strategies σ and π can be "simulated" by strategies σ' and π' in the game G_{Δ_C} so that the set of runs U is "projected" onto the set of runs U' in the play $G_{\Delta_C}(\sigma',\pi')$ where $\mathcal{P}(U) = \mathcal{P}(U')$. Then, it is shown that $\mathcal{P}(U') = 0$. This is because the strategy σ' is "sufficiently similar" to the strategy σ_T (see above), and hence the probability of visiting ε in $G_{\Delta_C}(\sigma',\pi')$ is 1. From this we get $\mathcal{P}(U') = 0$, because U' consists only of infinite runs, which cannot visit ε. The arguments are subtle and rely on several auxiliary technical observations. ∎

In other words, the non-emptiness of $\mathscr{A}$ is always certified by at least one witness of W, and hence each stochastic BPA game with a non-empty $\mathscr{A}$ can be made smaller by "cutting off" $Att(W)$.

The procedure which "cuts off" the symbols $Att(W)$ is not completely trivial. A naive idea of removing the symbols of $Att(W)$ together with the rules where they appear (this was used for the stochastic BPA game of Example 4.10) does not always work. This is illustrated in the following example:

Example 4.12. Consider a stochastic BPA game $\hat{\Delta} = (\{X,Y,Z,R\}, \hookrightarrow, (\{X\},\emptyset,\{Y,Z,R\}), Prob)$, where $X \hookrightarrow X$, $X \hookrightarrow Y$, $X \hookrightarrow ZY$, $Y \overset{1}{\hookrightarrow} Y$, $Z \overset{1/2}{\hookrightarrow} X$, $Z \overset{1/2}{\hookrightarrow} R$, $R \overset{1}{\hookrightarrow} R$, and $T_{\hat{\Gamma}} = \{R\}$. The game is initiated in X. We have that $\mathscr{A} = \{Y\}$ (observe that $X,Z,R \in [T_\varepsilon]^{=1}_{\Box}$, because the strategy σ of player $\Box$ which always selects the rule $X \hookrightarrow ZY$ is (=1)-winning). We have that $C = W = Att(W) = \{Y\}$. If we remove Y together with all rules where Y appears, we obtain the game $\Delta' = (\{X,Z,R\}, \hookrightarrow, (\{X\},\emptyset,\{Z,R\}), Prob)$, where $X \hookrightarrow X$, $Z \overset{1/2}{\hookrightarrow} X$, $Z \overset{1/2}{\hookrightarrow} R$, $R \overset{1}{\hookrightarrow} R$. In the game Δ', X becomes a witness and hence the algorithm would incorrectly put X into $\mathscr{A}$.

Hence, the "cutting" procedure must be designed more carefully. Intuitively, we do not remove rules of the form $X \hookrightarrow ZY$ where $Y \in Att(W)$, but change them into $X \hookrightarrow Z'Y$, where the symbol Z' behaves like Z but it cannot reach ε. Thus, we obtain the following theorem:

Theorem 4.13. *The membership to $[T]^{=1}_{\Box}$ and $[T]^{=1}_{\Diamond}$ is decidable in **NP** ∩ **co-NP**. Both sets are effectively regular, and the associated finite-state automata are constructible by a deterministic polynomial-time algorithm with **NP** ∩ **co-NP** oracle. Further, there is a regular strategy $\sigma \in \Sigma$ that is (=1)-winning in every configuration of $[T]^{=1}_{\Box}$. Moreover, the strategy σ is constructible by a deterministic polynomial-time algorithm with **NP** ∩ **co-NP** oracle.*

Note that in Theorem 4.13, we do not claim the existence (and constructability) of a regular (=1)-winning strategy π for player $\Diamond$. Actually, such a strategy *does* effectively exist, but we only managed to find a relatively complicated and technical proof which, in our opinion, is of little

practical interest (we do not see any natural reason for implementing a strategy which guarantees that the probability of visiting T is strictly less than 1). Hence, this proof is not included in the paper.

5. Conclusions

We have solved the qualitative reachability problem for stochastic BPA games, retaining the same upper complexity bounds that have previously been established for termination [15]. One interesting question which remains unsolved is the decidability of the problem whether $val(\alpha) = 1$ for a given BPA configuration α (we can only decide whether player $\Box$ has a (=1)-winning strategy, which is sufficient but not necessary for $val(\alpha) = 1$). Another open problem is quantitative reachability for stochastic BPA games, where the methods presented in this paper seem insufficient.

References

[1] P. Abdulla, N.B. Henda, L. de Alfaro, R. Mayr, and S. Sandberg. Stochastic games with lossy channels. In *Proceedings of FoSSaCS 2008*, volume 4962 of *LNCS*, pages 35–49. Springer, 2008.
[2] C. Baier, N. Bertrand, and Ph. Schnoebelen. Verifying nondeterministic probabilistic channel systems against ω-regular linear-time properties. *ACM Trans. on Comp. Logic*, 9(1), 2007.
[3] T. Brázdil, V. Brožek, A. Kučera, and J. Obdržálek. Qualitative reachability in stochastic BPA games. Technical report FIMU-RS-2009-01, Faculty of Informatics, Masaryk University, 2009.
[4] T. Brázdil, V. Brožek, V. Forejt, and A. Kučera. Reachability in recursive Markov decision processes. In *Proceedings of CONCUR 2006*, volume 4137 of *LNCS*, pages 358–374. Springer, 2006.
[5] T. Brázdil, V. Brožek, V. Forejt, and A. Kučera. Reachability in recursive Markov decision processes. *I&C*, 206(5):520–537, 2008.
[6] K. Chatterjee, L. de Alfaro, and T. Henzinger. The complexity of stochastic Rabin and Streett games. In *Proceedings of ICALP 2005*, volume 3580 of *LNCS*, pages 878–890. Springer, 2005.
[7] K. Chatterjee, M. Jurdziński, and T. Henzinger. Simple stochastic parity games. In *Proceedings of CSL'93*, volume 832 of *LNCS*, pages 100–113. Springer, 1994.
[8] K. Chatterjee, M. Jurdziński, and T. Henzinger. Quantitative stochastic parity games. In *Proceedings of SODA 2004*, pages 121–130. SIAM, 2004.
[9] A. Condon. The complexity of stochastic games. *I&C*, 96(2):203–224, 1992.
[10] L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *JCSS*, 68:374–397, 2004.
[11] J. Esparza, A. Kučera, and S. Schwoon. Model-checking LTL with regular valuations for pushdown systems. *I&C*, 186(2):355–376, 2003.
[12] K. Etessami, D. Wojtczak, and M. Yannakakis. Recursive stochastic games with positive rewards. In *Proceedings of ICALP 2008, Part I*, volume 5125 of *LNCS*, pages 711–723. Springer, 2008.
[13] 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.
[14] K. Etessami and M. Yannakakis. Efficient qualitative analysis of classes of recursive Markov decision processes and simple stochastic games. In *Proceedings of STACS 2006*, volume 3884 of *LNCS*, pages 634–645. Springer, 2006.
[15] K. Etessami and M. Yannakakis. Recursive concurrent stochastic games. In *Proceedings of ICALP 2006*, volume 4052 of *LNCS*, pages 324–335. Springer, 2006.
[16] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
[17] A. Maitra and W. Sudderth. Finitely additive stochastic games with Borel measurable payoffs. *Int. Jour. of Game Theory*, 27:257–267, 1998.
[18] D.A. Martin. The determinacy of blackwell games. *Journal of Symbolic Logic*, 63(4):1565–1581, 1998.
[19] W. Thomas. Infinite games and verification. In *Proceedings of CAV 2003*, volume 2725 of *LNCS*, pages 58–64. Springer, 2003.
[20] N. Vieille. Stochastic games: Recent results. *Handbook of Game Theory*, pages 1833–1850, 2002.
[21] I. Walukiewicz. A landscape with games in the background. In *Proceedings of LICS 2004*, pages 356–366. IEEE, 2004.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 219–230
www.stacs-conf.org

LOCALLY DECODABLE QUANTUM CODES

JOP BRIËT [1] AND RONALD DE WOLF [1]

[1] CWI. Kruislaan 413. 1098SJ Amsterdam. The Netherlands.
E-mail address: {jop.briet,rdewolf}@cwi.nl

Abstract. We study a quantum analogue of locally decodable error-correcting codes. A q-query *locally decodable quantum code* encodes n classical bits in an m-qubit state, in such a way that each of the encoded bits can be recovered with high probability by a measurement on at most q qubits of the quantum code, even if a constant fraction of its qubits have been corrupted adversarially. We show that such a quantum code can be transformed into a *classical* q-query locally decodable code of the same length that can be decoded well on average (albeit with smaller success probability and noise-tolerance). This shows, roughly speaking, that q-query quantum codes are not significantly better than q-query classical codes, at least for constant or small q.

1. Introduction

Locally decodable codes (LDCs) have received much attention in the last decade. They are error-correcting codes that encode n bits into m bits, with the usual error-correcting properties, and the additional feature that any one of the n encoded bits can be recovered (with high probability) by a randomized decoder that queries at most q bits in the codeword, for some small q. In other words, to decode small parts of the encoded data, we only need to look at a small part of the codeword instead of "unpacking" the whole thing. Precise definitions will be given in the next sections. Such codes are potentially useful in their own right (think of decoding small pieces from a large encoded library), and also have a variety of applications in complexity theory and cryptography. For instance, it is well known that they can be turned into private information retrieval schemes and vice versa. For further details about such connections, we refer to Trevisan's survey [Tre04] and the references therein.

The most interesting question about LDCs is the tradeoff between their length m and the number of queries q. The former measures the space efficiency of the code, while the latter measures the efficiency of decoding. The larger we make q, the smaller we can

1998 ACM Subject Classification: E1, E4, F1.

Key words and phrases: Data structures, locally decodable codes, quantum computing.

Jop Briët is partially supported by a Vici grant from the Netherlands Organization for Scientific Research (NWO). Ronald de Wolf is partially supported by Veni and Vidi grants from the Netherlands Organization for Scientific Research (NWO). Both authors are partially supported by the European Commission under the Integrated Project Qubit Applications (QAP) funded by the IST directorate as Contract Number 015848.

make m. On one extreme, if we allow $q = \text{polylog}(n)$ queries, the codelength m can be made polynomial in n [BFLS91]. On the other extreme, for $q = 1$ and sufficiently large n, LDCs do not exist at all [KT00]. For $q = 2$ they do exist but need exponential length, $m = \exp(n)$ [KW04]. Between these two extremes, interesting but hard questions persist. In particular, we know little about the length of LDCs with constant $q > 2$. The best upper bounds are based on Yekhanin's construction [Yek07]. Hhe gives 3-query LDCs with length $m = \exp(n^{1/t})$ for every *Mersenne prime* $p = 2^t - 1$. (The largest known Mersenne prime has $t = 32582657$, but it has been conjectured that there are infinitely many.) Recently, Efremenko [Efr08] used Yekhanin's basic underlying combinatorial structure to obtain, for integer $r \geq 2$, 2^r-query LDCs with $m = \exp(\exp(O(\log n \log\log^{r-1} n)^{1/r}))$, and 3-query LDCs with $m = \exp(\exp(O(\sqrt{\log n \log\log n})))$. On the lower bound side, the best we know for $q > 2$ is $m = \Omega\left((n/\log n)^{1+1/(\lceil q/2\rceil - 1)}\right)$ (for fixed success probability and noise rate) [KT00, KW04, Woo06]. For $q = 3$ and $q = 4$, this is slightly less than n^2.

Interestingly, the best known lower bounds were obtained using tools from quantum information theory. It is thus a natural question to consider also the potential *positive* effects of quantum: can we construct shorter q-query LDCs by somehow harnessing the power of quantum states and quantum algorithms? There are two natural ways to generalize locally decodable codes to the quantum world:

- We can keep the code classical, but allow q quantum queries. This means we can query positions of the codeword in quantum superposition, and process the results using quantum circuits. This approach was investigated in [KW04]. A q-query quantum decoder can simulate a $2q$-query classical decoder with high success probability, and this simulation can be made exact if the classical decoder took the parity of its $2q$ bits. This implies for instance that Efremenko's 3-query LDC can be decoded by only 2 quantum queries. In contrast, we know that every 2-query LDC needs length $\exp(n)$. Allowing quantum queries thus results in very large savings in m when we consider a fixed number of queries q.
- We can also make the code itself quantum: instead of encoding an n-bit x into an m-bit string $C(x)$, we could encode it into an m-qubit state $Q(x)$. A q-query decoder for such a code would select up to q qubits of the state $Q(x)$, and make a 2-outcome measurement on those qubits to determine its output. In this case our notion of noise also needs to be generalized: instead of up to δm bitflip-errors, we allow any set of up to δm qubits of $Q(x)$ to be arbitrarily changed.[1]

[1]While a classical LDC can be reused as often as we want, a quantum code has the problem that a measurement made to predict one bit changes the state, so predicting another bit based on the changed state may give the wrong results. However, if the error probability is small then the changes incurred by each measurement will be small as well, and we can reuse the code many times with reasonable confidence. Another issue is that more general decoders could be allowed. For instance, we could consider allowing any quantum measurement on the m-qubit state that can be written as a linear combination of m-qubit Pauli-matrices that have support on at most q positions. This is potentially stronger than what we do now (it is an interesting open question whether it is really stronger). However, we feel this is a somewhat unnatural formalization of the idea that a measurement should be localized to at most q qubits. Our current set-up, where we classically select up to q positions and then apply an arbitrary quantum measurement to those q qubits, seems more natural. Similarly, more general noise operators can be defined, but we will not consider these here.

Our results. In this paper we investigate the second kind of code, which we call a "q-query locally decodable quantum code", or q-query LDQC. The question is whether the ability to encode our n bits into a *quantum* state enables us to make codes much shorter. There are some small examples where quantum encodings achieve things that are impossible for classical encodings. Ambainis et al. [ANTV02] give an encoding of 2 classical bits into 1 qubit, such that each of the bits—though not both simultaneously—can be recovered from the qubit with success probability 0.85. They even cite an example due to Chuang where 3 bits are encoded into 1 qubit, and each bit can be recovered with success probability 0.78. However, they also show that asymptotically large savings are not possible in their setting (explained in Section 3.4 below). Their setting, however, considers neither noise nor local decodability, and hence does not answer our question about locally decodable codes: can LDCs be made significantly shorter if we allow quantum encodings?

Our main result is a negative answer to this question: essentially it says that q-query LDQCs can be turned into classical q-query LDCs of the same length, with some deterioration in their other parameters. The precise statement of this result (Corollaries 5.3 and 5.4) is a little bit dirty. We obtain a cleaner statement for so-called "smooth (quantum) codes", which have the property that they query the codewords fairly uniformly. These smooth (quantum) codes can be converted into LD(Q)Cs and vice versa. For these, the precise statement is as follows (Theorem 5.1). Suppose we are given a smooth quantum code of m qubits from which we can recover (with success probability at least $1/2 + \varepsilon$) each bit x_i of the encoded n-bit string x, while only looking at q qubits of the state. Let μ be a distribution on the n-bit inputs. Then we can construct a *randomized* classical code R of the same length (for each x, the "codeword" $R(x)$ is a *distribution* over m-bit strings) from which we can recover each x_i with μ-average success probability at least $1/2+\varepsilon/4^{q+1}$, while only looking at q bits of the codeword. Thus a q-query quantum code is turned into a q-query classical code of the same length, at the expense of reducing the advantage of the algorithm (over random guessing) by a factor of roughly 4^q.[2]

For those who do not like the idea of encoding x into a *distribution* $R(x)$, we can turn the randomized code R into a *deterministic* code C, where $C(x)$ is a fixed m-bit codeword instead of a distribution, at the expense of correctly decoding only a constant fraction of all indices i instead of all n of them (Corollary 5.2). Since all known lower bounds on LDCs also apply to randomized classical codes that work well under a uniform distribution on the n-bit strings, those lower bounds carry over to LDQCs. In particular we obtain as corollaries of our result:

- For sufficiently large n, 1-query LDQCs do not exist for any m (from [KT00]).[3]
- 2-query LDQCs need $m = \exp(n)$ (from [KW04]).
- For fixed q, q-query LDQCs need $m = \Omega\left((\frac{n}{\log n})^{1+1/(\lceil q/2\rceil-1)}\right)$ (from [KW04]).

Techniques. Our main technique is to apply to the m qubits of the quantum code a randomly selected sequence of m Pauli measurements. The randomized "codeword" $R(x)$ will be the probability distribution on m-bit outcomes resulting from such a measurement on the quantum state $Q(x)$. The meat of our proof is to show that there is a choice of measurements that roughly preserves correct decodability for all i.

[2] Oded Regev showed us how to improve this to 3^q, but we won't give the details of his improvement here.

[3] Actually, this result can more easily be shown directly, by combining Katz and Trevisan's proof for classical codes with the quantum random access code lower bound mentioned below in Section 3.4.

2. Preliminaries

We write $[n]$ for the set $\{1, \ldots, n\}$. We use $\mathcal{P}(S)$ to denote the set of all probability distributions (or random variables) on set S. If z is distributed according to the distribution of a random variable Z, we write $z \sim Z$. Probabilities and expectations with a subscript '$i \in S$' should be read as taken over a uniformly random $i \in S$. We give a brief overview of quantum mechanics here, see [NC00] for more.

Quantum states. In quantum mechanics, a physical system is mathematically represented by a complex Hilbert space. A d-dimensional complex Hilbert space consists of all d-dimensional vectors with complex entries, endowed with the standard inner product. The *state* of a physical system is in turn represented by a density operator (a positive semidefinite linear operator with trace 1) acting on a Hilbert space. We use $\mathcal{B}_+^1(\mathcal{H}_d)$ to denote the set of all density operators on a d-dimensional complex Hilbert space. Two-dimensional Hilbert spaces are called *qubits*. An n-qubit state is a density operator on the tensor product of n qubits (a 2^n-dimensional Hilbert space).

Measurements. The most general k-outcome *measurement* on a physical system is defined as a set $\{A_1, \ldots, A_k\}$ of k positive semidefinite matrices that satisfy $\sum_{i=1}^k A_i = I$. The probability that the measurement of a system in a state ρ yields the i'th outcome is $\mathrm{Tr}(A_i\rho)$. Hence, the measurement yields a random variable $A(\rho)$ with $\Pr[A(\rho) = i] = \mathrm{Tr}(A_i\rho)$. With a measurement that has outcomes $+1$ and -1 (and corresponding operators A^+ and A^-) we associate an operator $A = A^+ - A^-$. The expected value of this measurement on a state ρ is then $\mathrm{Tr}(A\rho)$. Note that this equals the difference between the probabilities of outcomes $+1$ and -1, respectively.

Pauli matrices. The one-qubit Pauli operators are given by $I = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $X = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $Y = \left(\begin{smallmatrix} 0 & -i \\ i & 0 \end{smallmatrix}\right)$ and $Z = \left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$. For integer $k \geq 1$, the set of k-qubit Pauli operators is $\mathcal{P}_k := \{I, X, Y, Z\}^{\otimes k}$. These 4^k matrices form an orthonormal basis for the space of all $2^k \times 2^k$ complex matrices endowed with the inner product $\langle A, B\rangle = \frac{1}{2^k}\mathrm{Tr}(A^\dagger B)$. Each Pauli operator $S \in \mathcal{P}_k$ has a unique decomposition $S = S^+ - S^-$, with S^+ and S^- orthogonal projectors that satisfy $S^+ + S^- = I$. For this reason we associate a unique two-outcome measurement $\{S^+, S^-\}$ with each such S. A Pauli measurement $S \in \mathcal{P}_k$ of a k-qubit state ρ yields a ± 1-valued random variable $S(\rho)$ with expected value $\mathrm{Tr}(S\rho)$. However, we can also view $S \in \mathcal{P}_k$ as k separate one-qubit Pauli measurements, to be applied to the k qubits of the state, respectively. When viewed in this way, the result of measuring ρ is a k-bit random variable, i.e., a probability distribution on $\{\pm 1\}^k$. The product of those k bits equals the ± 1-valued random variable $S(\rho)$ mentioned before.

Super-operators. A super-operator is a mathematical representation of the most general transformation of a quantum state allowed by the laws of quantum mechanics. A super-operator $\mathcal{E}$ can be defined by a finite set $\{E_1, \ldots, E_k\}$ of linear operators (known as Kraus operators) that satisfy $\sum_{i=1}^k E_i^\dagger E_i = I$. The corresponding operation on a state ρ yields another density operator, $\mathcal{E}(\rho) = \sum_{i=1}^k E_i \rho E_i^\dagger$. This $\mathcal{E}(\rho)$ may act on a Hilbert space of a possibly different dimension, though we will not need that here. We say that $\mathcal{E}$ "acts trivially" on, say, the first qubit of the state if all its Kraus operators have the form $E_i = I \otimes E_i'$ for some E_i' acting on all but the first qubit.

3. Codes

3.1. Classical codes

For convenience, we write bits as ± 1 instead of 0/1. This way, if random variable $A \in \{\pm 1\}$ predicts bit x_i, we can write the bias of this prediction as an expectation: $\mathbb{E}[A \cdot x_i] = \Pr[A = x_i] - \Pr[A \neq x_i]$. Note that $\Pr[A = x_i] \geq 1/2 + \varepsilon$ iff $\mathbb{E}[A \cdot x_i] \geq 2\varepsilon$.

We start with classical codes. The formal definition of a locally decodable code involves a decoder $\mathcal{A}$ that receives input $i \in [n]$ and oracle access to a string $y \in \{\pm 1\}^m$, usually written as a superscript to $\mathcal{A}$. This y will be a codeword $C(x) \in \{\pm 1\}^m$ corrupted by some "error string" $E \in \{\pm 1\}^m$, which negates some of the bits of $C(x)$ (below, $C(x) \circ E$ denotes the entry-wise product of the two m-bit vectors $C(x)$ and E). The oracle "queries index $j \in [m]$" if it reads the j'th bit of y.

Definition 3.1 (LDC)**.** A function $C : \{\pm 1\}^n \to \{\pm 1\}^m$ is a (q, δ, ε)-*LDC* if there exists a probabilistic oracle algorithm $\mathcal{A}$ such that

(1) For every $x \in \{\pm 1\}^n$, every $i \in [n]$, and every $E \in \{\pm 1\}^m$ with at most δm -1's, we have $\Pr[\mathcal{A}^{C(x) \circ E}(i) = x_i] \geq 1/2 + \varepsilon$, where the probability is taken over the internal coin tosses of $\mathcal{A}$.
(2) $\mathcal{A}$ queries at most q indices of y. Queries are made non-adaptively, meaning that the indices to be queried are all selected before the querying starts.

An $\mathcal{A}$ satisfying the above is called a (q, δ, ε)-*local decoder* for C.

Since any δm indices can be corrupted, a local decoder must query the indices fairly uniformly. Otherwise, an adversary could choose to corrupt the most queried part of the code and ruin the decoder's success probability. Motivated by this property, Katz and Trevisan [KT00] defined a variation of LDCs called a *smooth codes*, defined only for uncorrupted codewords.

Definition 3.2 (Smooth code)**.** A function $C : \{\pm 1\}^n \to \{\pm 1\}^m$ is a (q, c, ε)-*smooth code* if there exists a probabilistic oracle algorithm $\mathcal{A}$ such that:

(1) For every $x \in \{\pm 1\}^n$ and $i \in [n]$, we have $\Pr[\mathcal{A}^{C(x)}(i) = x_i] \geq 1/2 + \varepsilon$.
(2) For every $i \in [n]$ and $j \in [m]$, we have $\Pr[\mathcal{A}^{(\cdot)}(i) \text{ queries index } j] \leq c/m$.
(3) $\mathcal{A}$ queries at most q indices (non-adaptively).

An $\mathcal{A}$ satisfying the above is called a (q, c, ε)-smooth decoder for C.

Katz and Trevisan showed that LDCs and smooth codes are essentially equivalent, in the sense that a decoder for one can be transformed into a decoder for the other. We prove the same for quantum codes in Section 3.3, using essentially their proof.

3.2. Randomized codes

Here we define our first generalization, incorporating randomness into the definition of the code. A *randomized locally decodable code* maps $\{\pm 1\}^n$ to *random variables* over $\{\pm 1\}^m$ (rather than fixed codewords), such that any x_i can be decoded well using a constant number of queries, even if up to δm indices are corrupted.

Definition 3.3 (Randomized LDC)**.** A function $R : \{\pm 1\}^n \to \mathcal{P}(\{\pm 1\}^m)$ is a (q, c, ε)-*randomized LDC* if there exists a probabilistic oracle algorithm $\mathcal{A}$ such that:

(1) For every $x \in \{\pm 1\}^n$, every $i \in [n]$, and every $E \in \{\pm 1\}^m$ with at most δm -1's, we have $\Pr[\mathcal{A}^{R(x)\circ E}(i) = x_i] \geq 1/2 + \varepsilon$, where the probability is taken over the internal coin tosses of $\mathcal{A}$ as well as the distribution $R(x)$.
(2) $\mathcal{A}$ queries at most q indices (non-adaptively).

Similarly, we define a *randomized smooth code*:

Definition 3.4 (Randomized smooth code)**.** A function $R : \{\pm 1\}^n \to \mathcal{P}(\{\pm 1\}^m)$ is a (q, c, ε)*-randomized smooth code* if there exists a probabilistic oracle algorithm $\mathcal{A}$ such that:

(1) For every $x \in \{\pm 1\}^n$ and every $i \in [n]$, $\Pr[\mathcal{A}^{R(x)}(i) = x_i] \geq 1/2 + \varepsilon$.
(2) For every $i \in [n]$, and every $j \in [m]$, $\Pr[\mathcal{A}^{(\cdot)}(i) \text{ queries index } j] \leq c/m$.
(3) $\mathcal{A}$ queries at most q indices (non-adaptively).

It will be convenient to also have a version of these codes that are only required to work well on average, instead of for all x:

Definition 3.5 (μ-average codes)**.** Let μ be a distribution on $\{\pm 1\}^n$. A function $C : \{\pm 1\}^n \to \{\pm 1\}^m$ is a μ*-average* (q, δ, ε)*-LDC* if Definition 3.1 holds with the first clause replaced by:

(1) For every $i \in [n]$ and $E \in \{\pm 1\}^m$ with at most δm -1's, $\Pr_{x\sim\mu}[\mathcal{A}^{C(x)\circ E}(i) = x_i] \geq \frac{1}{2} + \varepsilon$.

Analogously, we define μ-average versions of smooth codes, randomized LDCs, and randomized smooth codes. For these codes, we assume without loss of generality that for each i and queried set $r \subseteq [m]$, the decoder $\mathcal{A}$ always uses the same function $f_{i,r} : \{\pm 1\}^q \to \{\pm 1\}$ to determine its output.

A μ-average randomized smooth code can actually be "derandomized" to a μ-average smooth code on a smaller number of bits:

Lemma 3.6. *Let $R : \{\pm 1\}^n \to \mathcal{P}(\{\pm 1\}^m)$ be a μ-average (q, c, ε)-randomized smooth code. Then there exists a μ-average $(q, c, \varepsilon/2)$-smooth code $C : \{\pm 1\}^n \to \{\pm 1\}^m$ for at least εn of the indices i (that is, a smooth code with μ-success probability at least $1/2 + \varepsilon/2$ for at least εn of the n indices).*

Proof. As a first step we will view R as a function to strings: there exists a random variable W (over some possibly infinite set $\mathcal{W}$) and a function $R : \{\pm 1\}^n \times \mathcal{W} \to \{\pm 1\}^m$ such that for every $x \in \{\pm 1\}^n$, the random variables $R(x, W)$ and $R(x)$ are the same. A decoder $\mathcal{A}$ for R also works for $R(\cdot, W)$, so we have bias $\mathbb{E}_{x\sim\mu, w\sim W}[\mathcal{A}^{R(x,w)}(i) \cdot x_i] \geq 2\varepsilon$ for every $i \in [n]$. For every $i \in [n]$ and $w \in \mathcal{W}$, define variables $X_{i,w} \in \{0, 1\}$, with

$$X_{i,w} = 1 \iff \mathbb{E}_{x\sim\mu}[\mathcal{A}^{R(x,w)}(i) \cdot x_i] \geq \varepsilon,$$

and $X_w := \sum_{i=1}^n X_{i,w}$. Using the definition of a μ-average randomized smooth code:

$$\begin{aligned} 2\varepsilon n &\leq \sum_{i=1}^n \mathbb{E}_{x\sim\mu, w\sim W}[\mathcal{A}^{R(x,w)}(i) \cdot x_i] = \mathbb{E}_{w\sim W}\left[\sum_{i=1}^n \mathbb{E}_{x\sim\mu}[\mathcal{A}^{R(x,w)}(i) \cdot x_i]\right] \\ &< \mathbb{E}_{w\sim W}[X_w + (n - X_w)\varepsilon] = \varepsilon n + (1 - \varepsilon)\mathbb{E}_{w\sim W}[X_w]. \end{aligned}$$

Hence $\mathbb{E}_{w\sim W}[X_w] \geq \varepsilon n$. Thus there exists a $w \in \mathcal{W}$ such that for at least εn of the n indices i, we have $\mathbb{E}_{x\sim\mu}[\mathcal{A}^{R(x,w)}(i) \cdot x_i] \geq \varepsilon$, equivalently, $\mathbb{E}_{x\sim\mu}[\Pr[\mathcal{A}^{R(x,w)}(i) = x_i]] \geq 1/2 + \varepsilon/2$. Defining the code $C(\cdot) := R(\cdot, w)$ gives the lemma. ∎

3.3. Quantum codes

Our second level of generalization brings quantum mechanics into the picture: now our code maps classical n-bit strings to m-qubit quantum states. Below, a "quantum oracle algorithm" is an algorithm $\mathcal{A}$ with oracle access to an m-qubit state ρ. This ρ could be a corrupted version of an m-qubit "codeword" $Q(x)$, obtained by applying some super-operator $\mathcal{E}$ to $Q(x)$. This $\mathcal{E}$ should only affect a δ-fraction of the m qubits. This error model generalizes the classical case: a classical error pattern $E \in \{\pm 1\}^m$ corresponds to a super-operator $\mathcal{E}$ that applies an X to the qubits at positions where E has a -1, and I to the positions where E has a $+1$. On input $i \in [n]$, the algorithm probabilistically selects a set $r \subseteq [m]$ of at most q indices of qubits of ρ, and applies a two-outcome measurement to the selected qubits with operators $A^+_{i,r}$ and $A^-_{i,r}$. As before, we will use "$\mathcal{A}^\rho(i)$" to denote the ± 1-valued random variable that is the output. We say that "$\mathcal{A}$ queries r", and "$\mathcal{A}$ queries index j" if j is in r. Note that such algorithms are non-adaptive by definition: the set of qubits r is selected before it is measured. We now define a *locally decodable quantum code* (LDQC) as follows:

Definition 3.7 (LDQC)**.** A function $Q : \{\pm 1\}^n \to \mathcal{B}^1_+(\mathcal{H}_{2^m})$ is a (q, δ, ε)-*LDQC* if there is a quantum oracle algorithm $\mathcal{A}$ s.t.:

(1) For every $x \in \{\pm 1\}^n$, every $i \in [n]$, and every super-operator $\mathcal{E}$ that acts non-trivially on at most δm qubits, we have $\Pr[\mathcal{A}^{\mathcal{E}(Q(x))}(i) = x_i] \geq 1/2 + \varepsilon$, where the probability is taken over the coin tosses and measurements in $\mathcal{A}$.
(2) $\mathcal{A}$ queries at most q indices (non-adaptively).

An $\mathcal{A}$ satisfying the above is called a (q, δ, ε)-local quantum decoder for Q.

LDQCs generalize randomized LDCs, because probability distributions are diagonal density operators. We can also establish a smoothness property for quantum codes:

Definition 3.8 (Smooth quantum code)**.** A function $Q : \{\pm 1\}^n \to \mathcal{B}^1_+(\mathcal{H}_{2^m})$ is a (q, c, ε)-*smooth quantum code* if there exists a quantum oracle algorithm $\mathcal{A}$ such that:

(1) For every $x \in \{\pm 1\}^n$ and every $i \in [n]$, we have $\Pr[\mathcal{A}^{Q(x)}(i) = x_i] \geq 1/2 + \varepsilon$.
(2) For every $i \in [n]$ and every $j \in [m]$, we have $\Pr[\mathcal{A}^{(\cdot)}(i)$ queries index j $] \leq c/m$.
(3) $\mathcal{A}$ queries at most q indices (non-adaptively).

An $\mathcal{A}$ satisfying the above is called a (q, c, ε)-smooth quantum decoder for Q.

As Katz and Trevisan [KT00] did for classical LDCs, we can establish a strong connection between LDQCs and smooth quantum codes. Either one can be used as the other, as the next theorems show. Analogues of these theorems also hold between randomized LDCs and randomized smooth codes, and between the μ-average versions of these codes.

Theorem 3.9. *Let $Q : \{\pm 1\}^n \to \mathcal{B}^1_+(\mathcal{H}_{2^m})$ be a (q, c, ε)-smooth quantum code. Then, as long as $\delta \leq \varepsilon/c$, Q is also a $(q, \delta, \varepsilon - \delta c)$-locally decodable quantum code.*

Proof. Let $\mathcal{A}$ be a (q, c, ε)-smooth quantum decoder for Q. Suppose we run it on $\mathcal{E}(Q(x))$ with at most δm corrupted qubits. The probability that $\mathcal{A}$ queries a specific qubit is at most c/m. Then by the union bound, the probability that $\mathcal{A}$ queries any of the corrupted qubits is at most $\delta mc/m = \delta c$. Hence $\mathcal{A}$ itself is also a $(q, \delta, \varepsilon - \delta c)$-local quantum decoder for Q. ∎

Theorem 3.10. *Let $Q : \{\pm 1\}^n \to \mathcal{B}_+^1(\mathcal{H}_{2^m})$ be a (q, δ, ε)-locally decodable quantum code. Then Q is also a $(q, q/\delta, \varepsilon)$-smooth quantum code.*

Proof. Let $\mathcal{A}$ be a (q, δ, ε)-local quantum decoder for Q. For each $i \in [n]$, let $p_i(j)$ be the probability that on input i, $\mathcal{A}$ queries qubit j. Let $H_i = \{j \mid p_i(j) > q/(\delta m)\}$. Then $|H_i| \leq \delta m$, because $\mathcal{A}$ queries no more than q indices. Let $\mathcal{B}$ be the quantum decoder that simulates $\mathcal{A}$, except that on input i it does not query qubits in H_i, but instead acts as if those qubits are in a completely mixed state. Then $\mathcal{B}$ does not measure any qubit j with probability greater than $q/(\delta m)$. Also, $\mathcal{B}$'s behavior on input i and $Q(x)$ is the same as $\mathcal{A}$'s behavior on input i and $\mathcal{E}(Q(x))$ that is obtained by replacing all qubits in H_i by completely mixed states. Since $\mathcal{E}$ acts non-trivially on at most $|H_i| \leq \delta m$ qubits, we have $\Pr[\mathcal{B}^{Q(x)}(i) = x_i] = \Pr[\mathcal{A}^{\mathcal{E}(Q(x))}(i) = x_i] \geq 1/2 + \varepsilon$. ∎

3.4. A weak lower bound from random access codes

We can immediately establish a weak lower bound on the length of LDQCs and smooth quantum codes by considering a *quantum random access code* (QRAC), introduced by Ambainis et al. [ANTV02].

Definition 3.11 (QRAC)**.** A function $Q : \{\pm 1\}^n \to \mathcal{B}_+^1(\mathcal{H}_{2^m})$ is an (n, m, ε)-*QRAC* if there exists a quantum oracle algorithm $\mathcal{A}$ such that for every $x \in \{\pm 1\}^n$ and $i \in [n]$, $\Pr[\mathcal{A}^{Q(x)}(i) = x_i] \geq 1/2 + \varepsilon$.

LDQCs and smooth quantum codes are QRACs with some additional properties, such as constraints on the way the qubits of the codeword are accessed. Hence the following well-known lower bound on the length of QRACs also holds for them.

Theorem 3.12 ([ANTV02, Nay99])**.** *Every (n, m, ε)-QRAC has $m \geq (1 - H(1/2 + \varepsilon))n$.*

4. Pauli decoding from disjoint subsets

In this section we consider a (q, c, ε)-smooth quantum code Q. Fix a distribution μ on $\{\pm 1\}^n$. We will show that there exists a sequence $S^* \in \mathcal{P}_m$ such that if the m qubits of $Q(x)$ are measured by the m Pauli measurements in S^*, then each x_i can be retrieved by querying only q bits of the m-bit measurement outcome $S^*(Q(x))$, in a very structured way. Specifically, we prove:

Theorem 4.1. *Let $Q : \{\pm 1\}^n \to \mathcal{B}_+^1(\mathcal{H}_{2^m})$ be a (q, c, ε)-smooth quantum code and μ be a distribution on $\{\pm 1\}^n$. Then, for sufficiently large n, there exists a sequence $S^* \in \mathcal{P}_m$, and for every $i \in [n]$ a set M_i of at least $\varepsilon m/(qc)$ disjoint sets $r \subseteq [m]$ (each of size at most q) with associated signs $a_{i,r} \in \{\pm 1\}$, such that*

$$\mathbb{E}_{x \sim \mu}\left[\frac{1}{|M_i|}\sum_{r \in M_i} \Pr\Big[a_{i,r}\prod_{j \in r} S_j^*(Q(x)) = x_i\Big]\right] \geq \frac{1}{2} + \frac{\varepsilon}{4^{q+1}}.$$

The proof consists of two parts. We start by constructing the sets M_i and then we show that decoding Q can be done by using only Pauli measurements. Putting these two observations together enables us to prove Theorem 4.1.

4.1. Decoding from disjoint subsets

First we obtain the sets M_i of disjoint q-sets that allow reasonable prediction of x_i.

Theorem 4.2 (modified from Lemma 4 in [KT00]). *Let $Q : \{\pm 1\}^n \to \mathcal{B}_+^1(\mathcal{H}_{2^m})$ be a (q, c, ε)-smooth quantum code with decoder $\mathcal{A}$, and μ a distribution on $\{\pm 1\}^n$. Then for every $i \in [n]$ there exists a set M_i of at least $\varepsilon m/(qc)$ disjoint sets $r \subseteq [m]$ (each of size at most q) satisfying $\Pr_{x\sim\mu}[\mathcal{A}^{Q(x)}(i) = x_i \mid \mathcal{A}^{(\cdot)}(i)$ queries $r] \geq 1/2 + \varepsilon/2$.*

Proof. Call a set $r \subseteq [m]$ "good for i" if it satisfies the above inequality. Define for every $i \in [n]$ a hypergraph $H_i = (V, E_i)$ with vertex-set $V = [m]$ and a set of hyperedges $E_i := \{e \mid e \text{ is good for } i\}$. A smooth quantum decoder $\mathcal{A}$ for Q "queries E_i" if $\mathcal{A}$ queries an $e \in E_i$. Let $p(e) := \Pr[\mathcal{A}^{(\cdot)}(i)$ queries $e]$. Then the probability that this decoder queries E_i is $p(E_i) := \sum_{e\in E_i} p(e)$. For all $e \notin E_i$ we have

$$\Pr[\mathcal{A}^{Q(x)}(i) = x_i \mid \mathcal{A}^{(\cdot)}(i) \text{ queries } e] < \frac{1}{2} + \frac{\varepsilon}{2}.$$

But since for every x and i, $\mathcal{A}$ decodes bit x_i with probability at least $1/2 + \varepsilon$, we have

$$\frac{1}{2} + \varepsilon \leq \Pr[\mathcal{A}^{Q(x)}(i) = x_i] < p(E_i) + (1 - p(E_i))(\frac{1}{2} + \frac{\varepsilon}{2}) = \frac{1}{2} + \frac{\varepsilon}{2} + p(E_i)(\frac{1}{2} - \frac{\varepsilon}{2}).$$

Hence $p(E_i) > \varepsilon/(1 - \varepsilon) \geq \varepsilon$. Since Q is *smooth*, we know that the probability that $\mathcal{A}$ queries an index j is $\sum_{e\in E_i | j\in e} p(e) = \Pr[\mathcal{A}^{(\cdot)}(i)$ queries $j] \leq c/m$.

Let M_i be a maximal set of disjoint hyperedges in H_i, and define the vertex set $T = \cup_{e\in M_i} e$. Note that T intersects each $e \in E_i$ (since otherwise M_i would not be maximal), and has at most $q|M_i|$ elements. We can now lower bound $|M_i|$ as follows:

$$\varepsilon < p(E_i) = \sum_{e\in E_i} p(e) \overset{(*)}{\leq} \sum_{j\in T} \sum_{e\in E_i | j\in e} p(e) \leq \frac{c|T|}{m} \leq \frac{cq|M_i|}{m},$$

where $(*)$ holds because each $e \in E_i$ is counted exactly once on the left-hand side, and at least once on the right-hand side (since T intersects each $e \in E_i$). ∎

4.2. Pauli decoding

In the second part of the proof of Theorem 4.1, we find the appropriate Pauli measurements. Recall that to decode x_i, a smooth quantum decoder first selects a set $r \subseteq [m]$ of at most q indices, and then applies some measurement with operators $A_{i,r}^+, A_{i,r}^-$ to determine its output. Let $A_{i,r} = A_{i,r}^+ - A_{i,r}^-$. Strictly speaking these operators act only on the qubits indexed by r, but we can view them as acting on the m-qubit state $Q(x)$ by tensoring them with $m - |r|$ identities. The difference between the probabilities of obtaining outcomes $+1$ and -1 is $\mathrm{Tr}(A_{i,r} \cdot Q(x))$. For every $i \in [n]$ and $r \in M_i$ we define the following bias:

$$B(i, r) := \mathbb{E}_{x\sim\mu}[\mathrm{Tr}(A_{i,r} \cdot Q(x)) \cdot x_i].$$

This measures how well the measurement outcome is correlated with x_i (with x weighted according to μ). By Theorem 4.2 we have $B(i, r) \geq \varepsilon$ for every $i \in [n]$ and each $r \in M_i$.

Since $\mathcal{P}_q$ is a basis for all $2^q \times 2^q$ complex matrices we can write

$$A_{i,r} = \sum_{S\in\mathcal{P}_q} \widehat{A_{i,r}}(S) S,$$

with $\widehat{A_{i,r}}(S) := \langle A_{i,r}, S\rangle = \frac{1}{2^q}\mathrm{Tr}(A_{i,r}\cdot S) \in [-1,1]$. We now have:

$$\varepsilon \leq B(i,r) = \sum_{S\in\mathcal{P}_q} \widehat{A_{i,r}}(S)\mathbb{E}_{x\sim\mu}[\mathrm{Tr}(S\cdot Q(x))\cdot x_i] \leq \sum_{S\in\mathcal{P}_q} \left|\mathbb{E}_{x\sim\mu}[\mathrm{Tr}(S\cdot Q(x))\cdot x_i]\right|. \quad (4.1)$$

Suppose we measure the r-qubits of $Q(x)$ with some $S\in\mathcal{P}_q$ and get outcome $b\in\{\pm 1\}$. The quantity $\mathbb{E}_{x\sim\mu}[\mathrm{Tr}(S\cdot Q(x))\cdot x_i]$ is the difference between $\Pr_{x\sim\mu}[b=x_i]$ and $\Pr_{x\sim\mu}[b\neq x_i]$. If we output b if this difference is nonnegative, and $-b$ otherwise, then we would predict x_i with bias

$$B'(i,S,r) := \left|\mathbb{E}_{x\sim\mu}[\mathrm{Tr}(S\cdot Q(x))\cdot x_i]\right|.$$

From Equation (4.1) we know that this bias is at least $\varepsilon/4^q$ for at least one "good" $S\in\mathcal{P}_q$. Hence, with some loss in success probability, we can decode Q by only using Pauli measurements. We now use a probabilistic argument to prove that a good sequence S^* of Pauli measurements exists, which is simultaneously good, for every $i\in[n]$, for most of the elements $r\in M_i$.

Proof (of Theorem 4.1). Suppose we let $\mathbf{S}\in\mathcal{P}_q$ be a random variable uniformly distributed over $\mathcal{P}_q$, and we use it to predict x_i as above. Then $B'(i,\mathbf{S},r)$ is a random variable in the interval $[0,1]$, with expectation

$$\mathbb{E}_{S\in\mathcal{P}_q}[B'(i,S,r)] = \frac{1}{4^q}\sum_{S\in\mathcal{P}_q}\left|\mathbb{E}_{x\sim\mu}[\mathrm{Tr}(S\cdot Q(x))\cdot x_i]\right| \geq \frac{\varepsilon}{4^q}.$$

Now we consider m-qubit Pauli measurements and replace all elements not in r with I's: for $S\in\mathcal{P}_m$ and $r\subseteq[m]$, let $S_{(r)}$ denote S with all its $m-|r|$ elements outside of r replaced by I. If we let $\mathbf{S}$ be uniform over $\mathcal{P}_m$, we get biases $B'(i,\mathbf{S}_{(r)},r)$ for each $r\in M_i$, each in $[0,1]$ and with expectation at least $\varepsilon/4^q$ (over the choice of $S_{(r)}$). But note that the random variables $B'(i,\mathbf{S}_{(r)},r)$ are independent for different $r\in M_i$, since the elements of M_i are disjoint. Hence the average bias over all $r\in M_i$,

$$B'(\mathbf{S},i) := \frac{1}{|M_i|}\sum_{r\in M_i} B'(i,\mathbf{S}_{(r)},r),$$

is the average of $|M_i|$ independent random variables, each in $[0,1]$ and with expectation at least $\varepsilon/4^q$. By a Chernoff bound[4] the probability that $B'(\mathbf{S},i)$ is much smaller than its expectation, is small:

$$\Pr_{S\in\mathcal{P}_m}\left[B'(S,i) < \frac{1}{2}\frac{\varepsilon}{4^q}\right] \leq \Pr_{S\in\mathcal{P}_m}\left[B'(S,i) < \frac{1}{2}\mathbb{E}[B'(S,i)]\right] \leq \exp\left(-\frac{|M_i|\varepsilon}{8\cdot 4^q}\right).$$

By Theorems 3.12 and 4.2 we may assume $|M_i| > 8\cdot 4^q\log(n)/\varepsilon$. It follows that for each $i\in[n]$, the above probability is less than $1/n$. Hence, the union bound gives

$$\Pr_{S\in\mathcal{P}_m}\left[\exists i \text{ s.t. } B'(S,i) < \frac{1}{2}\frac{\varepsilon}{4^q}\right] \leq \sum_{i=1}^{n}\Pr_{S\in\mathcal{P}_m}\left[B'(S,i) < \frac{1}{2}\frac{\varepsilon}{4^q}\right] < 1.$$

We can thus conclude that there exists an $S^*\in\mathcal{P}_m$ such that for every $i\in[n]$ we have

$$\frac{1}{|M_i|}\sum_{r\in M_i} B'(i,S^*_{(r)},r) \geq \frac{1}{2}\frac{\varepsilon}{4^q}.$$

[4]See Equation (7) in [HR90]. A small modification of their proof shows that this bound not only holds for independent 0/1-variables, but also for independent variables in the interval $[0,1]$.

This implies the statement of the theorem. ∎

5. Classical codes from quantum codes

Theorem 4.1 implies that if we measure all m indices of a smooth quantum quantum code Q with the elements of S^*, then we get distributions on $\{\pm 1\}^m$ that can be massaged to "codewords" $R(x)$ of a randomized smooth code:

Theorem 5.1. *Let $Q : \{\pm 1\}^n \to \mathcal{B}_+^1(\mathcal{H}_{2^m})$ be a (q, c, ε)-smooth quantum code. Then, for sufficiently large n, for every input distribution μ on $\{\pm 1\}^n$, there exists a μ-average $(q, qc/\varepsilon, \varepsilon/4^{q+1})$-randomized smooth code $R : \{\pm 1\}^n \to \mathcal{P}(\{\pm 1\}^m)$.*

Proof. We use Theorem 4.1. Let $R(x)$ be the distribution on $\{\pm 1\}^m$ obtained by measuring $Q(x)$ with S^*. We define a decoder $\mathcal{A}$ for R as follows: on input $i \in [m]$ and oracle $y \in \{\pm 1\}^m$, pick a set r from the set M_i uniformly at random, and return $a_{i,r} \prod_{j \in r} y_j$. It is straightforward to check that $\mathcal{A}$ is a μ-average $(q, qc/\varepsilon, \varepsilon/4^{q+1})$ decoder for R; in particular, since $\mathcal{A}$ picks r uniformly from a set of at least $\varepsilon m/(qc)$ disjoint sets, each index $j \in [m]$ has probability at most $qc/(\varepsilon m)$ of being queried. ∎

Together, Lemma 3.6 and Theorem 5.1, give the following "derandomization":

Corollary 5.2. *Let $Q : \{\pm 1\}^n \to \mathcal{B}_+^1(\mathcal{H}_{2^m})$ be a (q, c, ε)-smooth quantum code. Then, for sufficiently large n, for every distribution μ on $\{\pm 1\}^n$, there exists a $C : \{\pm 1\}^n \to \{\pm 1\}^m$ which is a μ-average $(q, qc/\varepsilon, \varepsilon/(2 \cdot 4^{q+1}))$-smooth code for at least $\varepsilon n/4^{q+1}$ of the n indices.*

Following the path through Theorems 3.10, 5.1, and the μ-average version of Theorem 3.9, we can turn an LDQC into a μ-average randomized LDC:

Corollary 5.3. *Let $Q : \{\pm 1\}^n \to \mathcal{B}_+^1(\mathcal{H}_{2^m})$ be a (q, δ, ε)-LDQC. Then, as long as $\delta' \leq \delta\varepsilon^2/(q^2 4^{q+1})$ and n is sufficiently large, for every distribution μ over $\{\pm 1\}^n$, there exists an $R : \{\pm 1\}^n \to \mathcal{P}(\{\pm 1\}^m)$ which is a μ-average $(q, \delta', \varepsilon/4^{q+1} - \delta' q^2/(\delta\varepsilon))$-randomized LDC.*

Going through Theorem 3.10, Corollary 5.2, and the μ-average version of Theorem 3.9 instead, we can also turn an LDQC into a μ-average LDC:

Corollary 5.4. *Let $Q : \{\pm 1\}^n \to \mathcal{B}_+^1(\mathcal{H}_{2^m})$ be a (q, δ, ε)-LDQC. Then, as long as $\delta' \leq \delta\varepsilon^2/(2q^2 4^{q+1})$ and n is sufficiently large, for every distribution μ over $\{\pm 1\}^n$, there exists a $C : \{\pm 1\}^n \to \{\pm 1\}^m$ which is a μ-average $(q, \delta', \varepsilon/(2 \cdot 4^{q+1}) - \delta' q^2/(\delta\varepsilon))$-LDC for at least $\varepsilon n/4^{q+1}$ of the n indices.*

6. Conclusion and open problems

We defined quantum generalizations of q-query LDCs in which q queries correspond to a measurement on q qubits of the m-qubit codeword. By a reduction to (classical) randomized smooth codes through a special sequence of Pauli measurements on an LDQC, we showed that the use of quantum systems for this type of encoding cannot provide much advantage in terms of length, at least for small q. An obvious open problem is reducing the gap between upper and lower bound on the length m of LDCs for fixed small number of queries q. Our results show that an upper bound for LDQCs would carry over to (μ-average) LDCs. This might perhaps be a way to improve the best known classical upper bounds on m.

Acknowledgments

We thank Harry Buhrman, Peter Høyer, Oded Regev, Falk Unger and Stephanie Wehner for useful discussions. JB is especially indebted to Peter Høyer for suggesting to turn an LDQC into an LDC via measurements.

References

[ANTV02] A. Ambainis, A. Nayak, A. Ta-Shma, and U. Vazirani. Dense quantum coding and quantum finite automata. *J. of ACM*, 49(4):496–511, 2002.

[BFLS91] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proc. of 23rd ACM STOC*, pages 21–31, 1991.

[CGKS98] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *J. of ACM*, 45(6):965–981, 1998.

[Efr08] K. Efremenko. 3-Query Locally Decodable Codes of Subexponential Length. Technical report, ECCC TR08–069, 2008.

[HR90] T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33(6):305–308, 1990.

[KT00] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proc. of 32nd ACM STOC*, pages 80–86, 2000.

[KW04] I. Kerenidis and R. de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *J. of Computer and System Sciences*, 69(3):395–420, 2004.

[Nay99] A. Nayak. Optimal lower bounds for quantum automata and random access codes. In *Proc. of 40th IEEE FOCS*, pages 369–376, 1999.

[NC00] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[Tre04] L. Trevisan. Some applications of coding theory in computational complexity. *Quaderni di Matematica*, 13:347–424, 2004.

[Woo06] D. Woodruff. New lower bounds for general locally decodable codes. Technical report, ECCC TR07–006, 2006.

[Yao77] A. C-C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. of 18th IEEE FOCS*, pages 222–227, 1977.

[Yek07] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *Proc. of 39th ACM STOC*, pages 266–274, 2007.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 231–242
www.stacs-conf.org

ENUMERATING HOMOMORPHISMS

ANDREI A. BULATOV [1] AND VÍCTOR DALMAU [2] AND MARTIN GROHE [3] AND DÁNIEL MARX [4]

School of Computing Science, Simon Fraser University, Burnaby, Canada
E-mail address: abulatov@cs.sfu.ca

Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain
E-mail address: victor.dalmau@tecn.upf.es

Institut für Informatik, Humboldt-Universität, Berlin, Germany
E-mail address: grohe@informatik.hu-berlin.de

Department of Computer Science and Information Theory, Budapest University of Technology and Economics, Budapest, Hungary
E-mail address: dmarx@cs.bme.hu

ABSTRACT. The homomorphism problem for relational structures is an abstract way of formulating constraint satisfaction problems (CSP) and various problems in database theory. The decision version of the homomorphism problem received a lot of attention in literature; in particular, the way the graph-theoretical structure of the variables and constraints influences the complexity of the problem is intensively studied. Here we study the problem of enumerating all the solutions with polynomial delay from a similar point of view. It turns out that the enumeration problem behaves very differently from the decision version. We give evidence that it is unlikely that a characterization result similar to the decision version can be obtained. Nevertheless, we show nontrivial cases where enumeration can be done with polynomial delay.

1. Introduction

Constraint satisfaction problems (CSP) form a rich class of algorithmic problems with applications in many areas of computer science. We only mention database systems, where CSPs appear in the guise of the conjunctive query containment problem and the closely related problem of evaluating conjunctive queries. It has been observed by Feder and Vardi [14] that as abstract problems, CSPs are homomorphism problems for relational structures. Algorithms for and the complexity of constraint satisfaction problems have been intensely studied (e.g. [20, 10, 4, 5]), not only for the standard decision problems but also optimization versions (e.g. [3, 22, 23, 24]) and counting versions (e.g. [6, 7, 8, 13]) of CSPs.

In this paper we study the *CSP enumeration problem*, that is, problem of computing all solutions for a given CSP instance. More specifically, we are interested in the question which structural restrictions on CSP instances guarantee tractable enumeration problems. "Structural restrictions"

The second author is supported by the MCyT through grants TIN2006-15387-C03-03 and TIN2007-68005-C04-03, and the program José Castillejo. Research of the fourth author is supported by the Magyary Zoltán Felsőoktatási Közalapítvány and the Hungarian National Research Fund (Grant Number OTKA 67651).

are restrictions on the structure induced by the constraints on the variables. Example of structural restrictions is "every variable occurs in at most 5 constraints" or "the constraints form an acyclic hypergraph.[1]" This can most easily be made precise if we view CSPs as homomorphism problems: Given two relational structures $\mathbb{A}, \mathbb{B}$, decide if there is a homomorphism from $\mathbb{A}$ to $\mathbb{B}$. Here the elements of the structure $\mathbb{A}$ correspond to the variables of the CSP and the elements of the structure $\mathbb{B}$ correspond to the possible values. Structural restrictions are restrictions on the structure $\mathbb{A}$. If $\mathcal{A}$ is a class of structures, then $\mathrm{CSP}(\mathcal{A}, -)$ denotes the restriction of the general CSP (or homomorphism problem) where the "left hand side" input structure $\mathbb{A}$ is taken from the class $\mathcal{A}$. $\mathrm{ECSP}(\mathcal{A}, -)$ denotes the corresponding enumeration problem: Given two relational structures $\mathbb{A} \in \mathcal{A}$ and $\mathbb{B}$, compute the set of all homomorphisms from $\mathbb{A}$ to $\mathbb{B}$. The enumeration problem is of particular interest in the database context, where we are usually not only interested in the question of whether the answer to a query is nonempty, but want to compute all tuples in the answer. We will also briefly discuss the corresponding *search* problem: Find a solution if one exists, denoted $\mathrm{SCSP}(\mathcal{A}, -)$.

It has been shown in [2] that $\mathrm{ECSP}(\mathcal{A}, -)$ can be solved in polynomial time if and only if the number of solutions (that is, homomorphisms) for all instances is polynomially bounded in terms of the input size and that this is the case if and only if the structures in the class $\mathcal{A}$ have bounded fractional edge cover number. However, usually we cannot expect the number of solutions to be polynomial. In this case, we may ask which conditions on $\mathcal{A}$ guarantee that $\mathrm{ECSP}(\mathcal{A}, -)$ has a polynomial delay algorithm. A *polynomial delay algorithm* for an enumeration problem is required to produce the first solution in polynomial time and then iteratively compute all solutions (each solution only once), leaving only polynomial time between two successive solutions. In particular, this guarantees that the algorithms computes all solutions in *polynomial total time*, that is, in time polynomial in the input size plus output size.

It is easy to see that $\mathrm{ECSP}(\mathcal{A}, -)$ has a polynomial delay algorithm if the class $\mathcal{A}$ has bounded tree width. It is also easy to see that there are classes $\mathcal{A}$ of unbounded tree width such that $\mathrm{ECSP}(\mathcal{A}, -)$ has a polynomial delay algorithm. It follows from our results that examples of such classes are the class of all grids or the class of all complete graphs with a loop on every vertex. It is known that the decision problem $\mathrm{CSP}(\mathcal{A}, -)$ is in polynomial time if and only if the cores of the structures in $\mathcal{A}$ have bounded tree width [17] (provided the arity of the constraints is bounded, and under some reasonable complexity theoretic assumptions). A *core* of a relational structure $\mathcal{A}$ is a minimal substructure $\mathcal{A}' \subseteq \mathcal{A}$ such that there is a homomorphism from $\mathcal{A}$ to $\mathcal{A}'$; minimality is with respect to inclusion. It is easy to see that all cores of a structure are isomorphic. Hence we usually speak of "the" core of a structure. Note that the core of a grid (and of any other bipartite graph with at least one edge) is a single edge, and the core of a complete graph with all loops present (and of any other graph with a loop) is a single vertex with a loop on it. The core of a complete graph with no loops is the graph itself. As a polynomial delay algorithm for an enumeration algorithms yields a polynomial time algorithm for the corresponding decision problem, it follows that $\mathrm{ECSP}(\mathcal{A}, -)$ can only have a polynomial delay algorithm if the cores of the structures in $\mathcal{A}$ have bounded tree width. Unfortunately, there are examples of classes $\mathcal{A}$ that have cores of bounded tree width, but for which $\mathrm{ECSP}(\mathcal{A}, -)$ has no polynomial delay algorithm unless P = NP (see Example 3.2).

Our main algorithmic results show that $\mathrm{ECSP}(\mathcal{A}, -)$ has a polynomial delay algorithm if the cores of the structures in $\mathcal{A}$ have bounded tree width and if, in addition, they can be reached in a sequence of "small steps." An *endomorphism* of a structure is a homomorphism of a structure to itself. A *retraction* is an endomorphism that is the identity mapping on its image. Every structure

[1]The other type of restrictions studied in the literature on CSP are "constraint language restrictions", that is, restrictions on the structure imposed by the constraint relations on the values. An example of a constraint language restriction is "all clauses of a SAT instance, viewed as a Boolean CSP, are Horn clauses".

has a retraction to its core. However, in general, the only way to map a structure to its core may be by collapsing the whole structure at once. As an example, consider a path with a loop on both endpoints. The core consists of a single vertex with a loop. (More precisely, the two cores are the two endpoints with their loops.) The only endomorphism of this structure to a proper substructure maps the whole structure to its core. Compare this with a path that only has a loop on one endpoint. Again, the core is a single vertex with a loop, but now we can reach the core by a sequence of retractions, mapping a path of length n to a subpath of length $n-1$ and then to a subpath of length $n-2$ et cetera. We prove that if $\mathcal{A}$ is a class of structures whose cores have bounded tree width and can be reached by a sequence of retractions each of which only moves a bounded number of vertices, then $\mathrm{ECSP}(\mathcal{A}, -)$ has a polynomial delay algorithm.

We also consider more general sequences of retractions or endomorphism from a structure to its core. We say that a sequence of endomorphisms from a structure $\mathbb{A}_0$ to a substructure $\mathbb{A}_1 \subset \mathbb{A}_0$, from $\mathbb{A}_1$ to a substructure $\mathbb{A}_2, \ldots$, to a structure $\mathbb{A}_n$ has *bounded width* if $\mathbb{A}_n$ and, for each $i \leq n$, the "difference between $\mathbb{A}_i$ and $\mathbb{A}_{i-1}$" has bounded tree width. We prove that if we are given a sequence of endomorphisms of bounded width together with the input structure $\mathbb{A}$, then we can compute all solutions by a polynomial delay algorithm. Unfortunately, in general we cannot compute such a sequence of endomorphisms efficiently. We prove that even for width 1 it is NP-complete to decide whether such a sequence exists.

Finally, we remark that our results are far from giving a complete classification of the classes $\mathcal{A}$ for which $\mathrm{ECSP}(\mathcal{A}, -)$ has a polynomial delay algorithm and those classes for which it does not. Indeed, we show that it will be difficult to obtain such a classification, because such a classification would imply a solution to the notoriously open *CSP dichotomy conjecture* of Feder and Vardi [14] (see Section 3 for details).

Due to space restrictions several proofs are omitted.

2. Preliminaries

Relational structures. A *vocabulary* τ is a finite set of *relation symbols* of specified arities. A *relational structure* $\mathbb{A}$ over τ consists of a finite set A called the *universe* of $\mathbb{A}$ and for each relation symbol $R \in \tau$, say, of arity r, an r-ary relation $R^{\mathbb{A}} \subseteq A^r$. Note that we require vocabularies and structures to be finite. A structure $\mathbb{A}$ is a *substructure* of a structure $\mathbb{B}$ if $A \subseteq B$ and $R^{\mathbb{A}} \subseteq R^{\mathbb{B}}$ for all $R \in \tau$. We write $\mathbb{A} \subseteq \mathbb{B}$ to denote that $\mathbb{A}$ is a substructure of $\mathbb{B}$ and $\mathbb{A} \subset \mathbb{B}$ to denote that $\mathbb{A}$ is a *proper* substructure of $\mathbb{B}$, that is, $\mathbb{A} \subseteq \mathbb{B}$ and $\mathbb{A} \neq \mathbb{B}$. A substructure $\mathbb{A} \subseteq \mathbb{B}$ is *induced* if for all $R \in \tau$, say, of arity r, we have $R^{\mathbb{A}} = R^{\mathbb{B}} \cap A^r$. For a subset $A \subseteq B$, we write $\mathbb{B}[A]$ to denote the induced substructure of $\mathbb{B}$ with universe A.

Homomorphisms. We often abbreviate tuples $(a_1, \ldots, a_k)$ by $\mathbf{a}$. If f is a mapping whose domain contains $a_1, \ldots, a_k$ we write $f(\mathbf{a})$ to abbreviate $(f(a_1), \ldots, f(a_k))$. A *homomorphism* from a relational structure $\mathbb{A}$ to a relational structure $\mathbb{B}$ is a mapping $\varphi : A \to B$ such that for all $R \in \tau$ and all tuples $\mathbf{a} \in R^{\mathbb{A}}$ we have $\varphi(\mathbf{a}) \in R^{\mathbb{B}}$. A *partial homomorphism* on $C \subseteq A$ to $\mathbb{B}$ is a homomorphism of $\mathbb{A}[C]$ to $\mathbb{B}$. It is sometimes useful when designing examples to exclude certain homomorphisms or endomorphisms. The simplest way to do that is to use unary relations. For example, if R is a unary relation and $(a) \in R^{\mathbb{A}}$ we say that a *has color* R. Now if $b \in B$ does not have color R then no homomorphism from $\mathbb{A}$ to $\mathbb{B}$ maps a to b.

Two structures $\mathbb{A}$ and $\mathbb{B}$ are *homomorphically equivalent* if there is a homomorphism from $\mathbb{A}$ to $\mathbb{B}$ and also a homomorphism from $\mathbb{B}$ to $\mathbb{A}$. Note that if structures $\mathbb{A}$ and $\mathbb{A}'$ are homomorphically

equivalent, then for every structure $\mathbb{B}$ there is a homomorphism from $\mathbb{A}$ to $\mathbb{B}$ if and only if there is a homomorphism from $\mathbb{A}'$ to $\mathbb{B}$; in other words: the instances $(\mathbb{A}, \mathbb{B})$ and $(\mathbb{A}', \mathbb{B})$ of the decision CSP are equivalent. However, the two instances may have vastly different sizes, and the complexity of solving the search and enumeration problems for them can also be quite different. Homomorphic equivalence is closely related to the concept of the core of a structure: A structure $\mathbb{A}$ is a *core* if there is no homomorphism from $\mathbb{A}$ to a proper substructure of $\mathbb{A}$. A core of a structure $\mathbb{A}$ is a substructure $\mathbb{A}' \subseteq \mathbb{A}$ such that there is a homomorphism from $\mathbb{A}$ to $\mathbb{A}'$ and $\mathbb{A}'$ is a core. Obviously, every core of a structure is homomorphically equivalent to the structure. We observe another basic fact about cores:

Observation 2.1. Let $\mathbb{A}$ and $\mathbb{B}$ be homomorphically equivalent structures, and let $\mathbb{A}'$ and $\mathbb{B}'$ be cores of $\mathbb{A}$ and $\mathbb{B}$, respectively. Then $\mathbb{A}'$ and $\mathbb{B}'$ are isomorphic. In particular, all cores of a structure $\mathbb{A}$ are isomorphic. Therefore, we often speak of *the* core of $\mathbb{A}$.

Observation 2.2. It is easy to see that it is NP-hard to decide, given structures $\mathbb{A} \subseteq \mathbb{B}$, whether $\mathbb{A}$ is isomorphic to the core of $\mathbb{B}$. (For an arbitrary graph G, let $\mathbb{A}$ be a triangle and $\mathbb{B}$ the disjoint union of G with $\mathbb{A}$. Then $\mathbb{A}$ is a core of $\mathbb{B}$ if and only if G is 3-colorable.) Hell and Nešetřil [19] proved that it is co-NP-complete to decide whether a graph is a core.

Tree decompositions. A *tree decomposition* of a graph G is a pair (T, B), where T is a tree and B is a mapping that associates with every node $t \in V(T)$ a set $B_t \subseteq V(G)$ such that (1) for every $v \in V(G)$ the set $\{t \in V(T) | v \in B_t\}$ is connected in T, and (2) for every $e \in E(G)$ there is a $t \in V(T)$ such that $e \subseteq B_t$. The sets B_t, for $t \in V(T)$, are called the *bags* of the decomposition. It is sometimes convenient to have the tree T in a tree decomposition rooted; we always assume it is. The *width* of a tree decomposition (T, B) is $\max\{|B_t| \mid t \in V(T)\} - 1$. The *tree width* of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum of the widths of all tree decompositions of G.

We need to transfer some of the notions of graph theory to arbitrary relational structures. The *Gaifman graph* (also known as *primal graph*) of a relational structure $\mathbb{A}$ with vocabulary τ is the graph $G(\mathbb{A})$ with vertex set $\mathbb{A}$ and an edge between a and b if $a \neq b$ and there is a relation symbol $R \in \tau$, say, of arity r, and a tuple $(a_1, \ldots, a_r) \in R^{\mathbb{A}}$ such that $a, b \in \{a_1, \ldots, a_r\}$. We can now transfer graph-theoretic notions to relational structures. In particular, a subset $B \subseteq A$ is *connected* in a structure $\mathbb{A}$ if it is connected in $G(\mathbb{A})$. A *tree decomposition* of a structure $\mathbb{A}$ can simply be defined to be a tree-decomposition of $G(\mathbb{A})$. Equivalently, a tree decomposition of $\mathbb{A}$ can be defined directly by replacing the second condition in the definition of tree decompositions of graphs by (2') for every $R \in \tau$ and $(a_1, \ldots, a_r) \in R^{\mathbb{A}}$ there is a $t \in V(T)$ such that $\{a_1, \ldots, a_r\} \subseteq B_t$. A class $\mathcal{C}$ of structures has *bounded tree width* if there is a $w \in \mathbb{N}$ such that $\mathrm{tw}(\mathbb{A}) \leq w$ for all $\mathbb{A} \in \mathcal{C}$. A class $\mathcal{C}$ of structures has *bounded tree width modulo homomorphic equivalence* if there is a $w \in \mathbb{N}$ such that every $\mathbb{A} \in \mathcal{C}$ is homomorphically equivalent to a structure of tree width at most w.

Observation 2.3. A structure $\mathbb{A}$ is homomorphically equivalent to a structure of tree width at most w if and only if the core of $\mathbb{A}$ has tree width at most w.

The Constraint Satisfaction Problem. For two classes $\mathcal{A}$ and $\mathcal{B}$ of structures, the *Constraint Satisfaction Problem*, $\mathrm{CSP}(\mathcal{A}, \mathcal{B})$, is the following problem:

$\mathrm{CSP}(\mathcal{A}, \mathcal{B})$
Instance: $\mathbb{A} \in \mathcal{A}, \mathbb{B} \in \mathcal{B}$
Problem: Decide if there is a homomorphism from $\mathbb{A}$ to $\mathbb{B}$.

The CSP is a decision problem. The variation of it we study in this paper is the following enumeration problem:

$\mathrm{ECSP}(\mathcal{A},\mathcal{B})$
Instance: $\mathbb{A}\in\mathcal{A}$, $\mathbb{B}\in\mathcal{B}$
Problem: Output all the homomorphisms from $\mathbb{A}$ to $\mathbb{B}$.

We shall also refer to the search problem, $\mathrm{SCSP}(\mathcal{A},\mathcal{B})$, in which the goal is to find one solution to a CSP-instance or output 'no' if a solution does not exists.

If one of the classes $\mathcal{A}$, $\mathcal{B}$ is the class of all finite structures, then we denote the corresponding CSPs by $\mathrm{CSP}(\mathcal{A},-)$, $\mathrm{CSP}(-,\mathcal{B})$ (respectively, $\mathrm{ECSP}(\mathcal{A},-)$, $\mathrm{ECSP}(-,\mathcal{B})$, $\mathrm{SCSP}(\mathcal{A},-)$, $\mathrm{SCSP}(-,\mathcal{B})$).

The decision CSP has been intensely studied. If a class $\mathcal{C}$ of structures has bounded arity then $\mathrm{CSP}(\mathcal{C},-)$ is solvable in polynomial time if and only if $\mathcal{C}$ has bounded tree width modulo homomorphic equivalence [17]. If the arity of $\mathcal{C}$ is not bounded, several quite general conditions on a class of structures have been identified that guarantee polynomial time solvability of $\mathrm{CSP}(\mathcal{C},-)$, see, e.g.[16, 12, 18]. Problems of the form $\mathrm{CSP}(-,\mathcal{C})$ have been studied mostly in the case when $\mathcal{C}$ is 1-element. Problems of this type are sometimes referred to as *non-uniform*. It is conjectured that every non-uniform problem is either solvable in polynomial time or NP-complete (the so-called *Dichotomy Conjecture*) [14]. Although this conjecture is proved in several particular cases [20, 9, 10, 4], in its general form it is believed to be very difficult.

A search CSP is clearly no easier than the corresponding decision problem. While any non-uniform search problem $\mathrm{SCSP}(-,\mathcal{C})$ is polynomial time reducible to its decision version $\mathrm{CSP}(-,\mathcal{C})$ [11], nothing is known about the complexity of search problems $\mathrm{SCSP}(\mathcal{C},-)$ except the result we state in Section 3. Paper [25] provides some initial results on the complexity of non-uniform enumerating problems.

3. Tractable structures for enumeration

Since even an easy CSP may have exponentially many solutions, the model of choice for 'easy' enumeration problems is algorithms with polynomial delay [21]. An algorithm Alg is said to solve a CSP *with polynomial delay* (*WPD* for short) if there is a polynomial $p(n)$ such that, for every instance of size n, Alg outputs 'no' in a time bounded by $p(n)$ if there is no solution, otherwise it generates all solutions to the instance such that no solution is output twice, the first solution is output after at most $p(n)$ steps after the computation starts, and time between outputting two consequent solutions does not exceed $p(n)$.

If a class of relational structures $\mathcal{C}$ has bounded arity, the aforementioned result of Grohe [17] imposes strong restrictions on enumeration problems solvable WPD.

Observation 3.1. If a class of relational structures $\mathcal{C}$ with bounded arity does not have bounded tree width modulo homomorphic equivalence, then $\mathrm{ECSP}(\mathcal{C},-)$ is not WPD, unless P=NP.

Unlike for the decision version, the converse is not true: bounded tree width modulo homomorphic equivalence does not imply enumerability WPD.

Example 3.2. Let $\mathbb{A}_k$ be the disjoint union of a k-clique and a loop and let $\mathcal{A}=\{\mathbb{A}_k \mid k\geq 1\}$. Clearly, the core of each graph in $\mathcal{A}$ has bounded tree width (in fact, it is a single element), hence $\mathrm{CSP}(\mathcal{A},-)$ is polynomial-time solvable. For an arbitrary graph $\mathbb{B}$ without loops, let $\mathbb{B}'$ be the disjoint union of $\mathbb{B}$ and a loop. It is clear that there is always a trivial homomorphism

from $\mathbb{A}_k$ (for any $k \geq 1$) to $\mathbb{B}'$ that maps everything into the loop. There exist homomorphisms different from the trivial one if and only if $\mathbb{B}$ contains a k-clique. Thus if we are able to check in polynomial time whether there is a second homomorphism, then we are able to test if $\mathbb{B}$ has a k-clique. Therefore, although $\mathrm{CSP}(\mathcal{A}, -)$ and $\mathrm{SCSP}(\mathcal{A}, -)$ are polynomial-time solvable, a WPD enumeration algorithm for $\mathrm{ECSP}(\mathcal{A}, -)$ would imply P = NP.

It is not difficult to show that $\mathrm{ECSP}(\mathcal{C}, -)$ is enumerable WPD if $\mathcal{C}$ has bounded tree width. For space restrictions we do not include a direct proof and instead we derive it from a more general result in Section 4. Thus enumerability WPD has a different tractability criterion than the decision version, and this criterion lies somewhere between bounded tree width and bounded tree width modulo homomorphic equivalence. Thus in order to ensure that the solutions can be enumerated WPD, we have to make further restrictions on the way the structure can be mapped to its bounded tree width core. The main new definition of the paper requires that the core is reached by "small steps":

Let $\mathbb{A}$ be a relational structure with universe A. We say that $\mathbb{A}$ *has a sequence of endomorphisms of width* k if there are subsets $A = A_0 \supset A_1 \supset \ldots \supset A_n \neq \emptyset$ and homomorphisms $\varphi_1, \ldots, \varphi_n$ such that

(1) φ_i is a homomorphism from $\mathbb{A}[A_{i-1}]$ to $\mathbb{A}[A_i]$,
(2) $\varphi_i(A_{i-1}) = A_i$ for $1 \leq i \leq n$;
(3) if G is the primal graph of $\mathbb{A}$, then the tree width of $G[A_i \setminus A_{i+1}]$ is at most k for every $0 \leq i < n$;
(4) the structure induced by A_n has tree width at most k.

In Section 4, we show that enumeration for $(\mathbb{A}, \mathbb{B})$ can be done WPD if a sequence of bounded width endomorphisms for $\mathbb{A}$ is given in the input. Unfortunately, we cannot claim that $\mathrm{ECSP}(\mathcal{A}, -)$ can be done WPD if every structure in $\mathcal{A}$ has such a sequence, since we do not know how to find such sequences efficiently. In fact, as we show in Section 5, it is hard to check if a width-1 sequence exists for a given structure. Furthermore, we show a class $\mathcal{A}$ where every structure has a width-2 sequence, but $\mathrm{ECSP}(\mathcal{A}, -)$ cannot be done WPD, unless P = NP. This means that it is not possible to get around the problem of not being able to find the sequences (for example, by finding sequences with somewhat larger width or by constructing the sequence during the enumeration).

Thus having a bounded width sequence of endomorphisms is not the right tractability criterion. We then investigate a more restrictive notion, where the bound is not on the tree width of the difference of the layers but on the number of elements in the differences. However, in the rest of the section, we give evidence that enumeration problems solvable WPD cannot be characterized in simple terms relying on tree width. For instance, a description of search problems solvable in polynomial time would imply a description of non-uniform decision problems solvable in polynomial time. This is shown via an analogous result for the search version of the problem, which might be of independent interest. By $\mathbb{A} \oplus \mathbb{B}$ we denote the disjoint union of relational structures $\mathbb{A}$ and $\mathbb{B}$.

Lemma 3.3. *Let $\mathbb{B}$ be a relational structure, which is a core, and let $\mathcal{C}_\mathbb{B}$ be $\{\mathbb{A} \oplus \mathbb{B} \mid \mathbb{A} \to \mathbb{B}\}$. Then $\mathrm{CSP}(-, \mathbb{B})$ is solvable in polynomial time if and only if so is the problem $\mathrm{SCSP}(\mathcal{C}_\mathbb{B}, -)$.*

Proof. If the decision problem $\mathrm{CSP}(-, \mathbb{B})$ is solvable in polynomial time we can construct an algorithm that given an instance $(\mathbb{A}, \mathbb{C})$ of $\mathrm{CSP}(\mathcal{C}_\mathbb{B}, -)$ computes a solution in polynomial time. Indeed, as $\mathrm{CSP}(-, \mathbb{B})$ is solvable in polynomial time by the aforementioned result of [11] it is also polynomial time to find a homomorphism from a given structure to $\mathbb{B}$ provided one exists. If $\mathbb{A} \in \mathcal{C}_\mathbb{B}$ such a homomorphism φ exists by the definition of $\mathcal{C}_\mathbb{B}$. So our algorithms, first, finds some homomorphism φ. Then it decides by brute force whether or not there exists a homomorphism φ' from $\mathbb{B}$ to $\mathbb{C}$ (note

that this can be done in polynomial time for every fixed $\mathbb{B}$). If such a homomorphism does not exist then we can certainly guarantee that there is no homomorphism from $\mathbb{A}$ to $\mathbb{C}$. Otherwise we obtain a required homomorphism ψ as follows: Let $\psi(a) = \varphi'(a)$ for $a \in \mathbb{B}$, and $\psi(a) = \varphi' \circ \varphi(a)$ for $a \in \mathbb{A}$.

Conversely, assume that we have an algorithm Alg that finds a solution of any instance of $\mathrm{CSP}(\mathcal{C}_{\mathbb{B}}, -)$ in polynomial time, say, $p(n)$. We construct from it an algorithm that solves $\mathrm{CSP}(-, \mathbb{B})$. Given an instance $(\mathbb{A}, \mathbb{B})$ of $\mathrm{CSP}(-, \mathbb{B})$ we call algorithm Alg with input $\mathbb{A} \oplus \mathbb{B}$ and $\mathbb{B}$. Additionally we count the number of steps performed by Alg in such a way that we stop if Alg has not finished in $p(n)$ steps. If Alg produces a correct answer then we have to be able to obtain from it a homomorphism from $\mathbb{A}$ to $\mathbb{B}$. If Alg's answer is not correct or the clock reaches $p(n)$ steps we know that Alg failed. The only possible reason for that is that $\mathbb{A} \oplus \mathbb{B}$ does not belong to $\mathcal{C}_{\mathbb{B}}$, which implies that $\mathbb{A}$ is not homomorphic to $\mathbb{B}$. ∎

In what follows we transfer this result to enumeration problems. Let $\mathcal{A}$ be a class of relational structures. The class $\mathcal{A}'$ consists of all structures built as follows: Take $\mathbb{A} \in \mathcal{A}$ and add to it $|\mathbb{A}|$ independent vertices.

Lemma 3.4. *Let $\mathcal{A}$ be a class of relational structures. Then $\mathrm{SCSP}(\mathcal{A}, -)$ is solvable in polynomial time if and only if $\mathrm{ECSP}(\mathcal{A}', -)$ is solvable WPD.*

Proof. If $\mathrm{ECSP}(\mathcal{A}, -)$ is enumerable WPD, then for any structure $\mathbb{A}' \in \mathcal{A}'$ it takes time polynomial in $|\mathbb{A}'|$ to find the first solution. Since $\mathbb{A}'$ is only twice of the size of the corresponding structure $\mathbb{A}$, it takes only polynomial time to solve $\mathrm{SCSP}(\mathcal{A}, -)$.

Conversely, given a structure $\mathbb{A}' = \mathbb{A} \cup I \in \mathcal{A}'$, where $\mathbb{A} \in \mathcal{A}$ and I is the set of independent elements, and any structure $\mathbb{B}$. The first homomorphism from $\mathbb{A}'$ to $\mathbb{B}$ can be found in polynomial time, since $\mathrm{SCSP}(\mathcal{A}, -)$ is polynomial time solvable and the independent vertices can be mapped arbitrarily. Let the restriction of this homomorphism onto $\mathbb{A}$ be φ. Then while enumerating all possible $|\mathbb{B}|^{|\mathbb{A}|}$ extensions of φ we buy enough time to enumerate all homomorphisms from $\mathbb{A}$ to $\mathbb{B}$ using brute force. ∎

4. Sequence of bounded width endomorphisms

In this section we show that for every fixed k, all the homomorphisms from $\mathbb{A}$ to $\mathbb{B}$ can be enumerated with polynomial delay if a sequence of width k endomorphisms of $\mathbb{A}$ is given in the input. Given a sequence $A_0, \ldots, A_n$ and $\varphi_1, \ldots, \varphi_n$ as in the definition of a sequence of width k endomorphisms, we denote $\mathbb{A}[A_i]$ by $\mathbb{A}_i$.

We will enumerate the homomorphisms from $\mathbb{A}$ to $\mathbb{B}$ by first enumerating the homomorphisms from $\mathbb{A}_n$, $\mathbb{A}_{n-1}$, ... to $\mathbb{B}$ and then transforming them to homomorphisms from $\mathbb{A}$ to $\mathbb{B}$ using the homomorphisms φ_i. We obtain the homomorphisms from $\mathbb{A}_i$ by extending the homomorphism from $\mathbb{A}_{i+1}$ to the set $A_i \setminus A_{i+1}$; Lemma 4.1 below will be useful for this purpose. In order to avoid producing a homomorphism multiple times, we need a delicate classification (see definitions of elementary homomorphisms and of the index of a homomorphism).

Lemma 4.1. *Let $\mathbb{A}, \mathbb{B}$ be relational structures and $X_1 \subseteq X_2 \subseteq A$ subsets, and let g_0 be a homomorphism from $\mathbb{A}[X_1]$ to B. For every fixed k, there is a polynomial-time algorithm* HOMOMORPHISM-EXT$(\mathbb{A}, \mathbb{B}, X_1, X_2, g_0)$ *that decides whether g_0 can be extended to a homomorphism from $\mathbb{A}[X_2]$ to $\mathbb{B}$, if the tree width of induced subgraph $G[X_2 \setminus X_1]$ of the Gaifman graph of $\mathbb{A}$ is at most k.*

The *index* of a homomorphism φ from $\mathbb{A}$ to $\mathbb{B}$ is the largest t such that φ can be written as $\varphi = \psi \circ \varphi_t \circ \ldots \circ \varphi_1$ for some homomorphism ψ from $\mathbb{A}_t$ to $\mathbb{B}$. In particular, if φ cannot be written as $\varphi = \psi \circ \varphi_1$, then the index of φ is 0. Observe that if the index of φ is at least t, then there is a unique ψ such that $\varphi = \psi \circ \varphi_t \circ \ldots \circ \varphi_1$: This follows from the fact that $\varphi_t \circ \ldots \circ \varphi_1$ is a surjective mapping from A to A_t, thus if ψ' and ψ'' differ on A_t, then $\psi' \circ \varphi_t \circ \ldots \circ \varphi_1$ and $\psi'' \circ \varphi_t \circ \ldots \circ \varphi_1$ differ on A. A homomorphism ψ from $\mathbb{A}_t$ to $\mathbb{B}$ is *elementary*, if it cannot be written as $\psi = \psi' \circ \varphi_{t+1}$. A homomorphism is *reducible* if it is not elementary.

Lemma 4.2. *If a homomorphism ψ from $\mathbb{A}_t$ to $\mathbb{B}$ is elementary, then $\varphi = \psi \circ \varphi_t \circ \ldots \circ \varphi_1$ has index exactly t. Conversely, if homomorphism φ from $\mathbb{A}$ to $\mathbb{B}$ has index t and can be written as $\varphi = \psi \circ \varphi_t \circ \ldots \circ \varphi_1$, then the homomorphism ψ from $\mathbb{A}_t$ to $\mathbb{B}$ is elementary.*

Lemma 4.2 suggests a way of enumerating all the homomorphisms from $\mathbb{A}$ to $\mathbb{B}$: for $t = 0, \ldots, n$, we enumerate all the elementary homomorphisms from $\mathbb{A}_t$ to $\mathbb{B}$, and for each such homomorphism ψ, we compute $\varphi = \psi \circ \varphi_t \circ \ldots \circ \varphi_1$. To this end, we need the following characterization of elementary homomorphisms:

Lemma 4.3. *A homomorphism ψ from $\mathbb{A}_t$ to $\mathbb{B}$ is reducible if and only if*

(1) *$\psi(x) = \psi(y)$ for every $x, y \in A_t$ with $\varphi_{t+1}(x) = \varphi_{t+1}(y)$, i.e., for every $z \in A_{t+1}$, $\psi(x)$ has the same value b_z for every x with $\varphi_{t+1}(x) = z$, and*
(2) *the mapping defined by $\psi'(z) := b_z$ is a homomorphism from $\mathbb{A}_{t+1}$ to $\mathbb{B}$.*

Lemma 4.3 gives a way of testing in polynomial time whether a given homomorphism ψ is elementary: we have to test whether one of the two conditions are violated. We state this in a more general form: we can test in polynomial time whether a partial mapping g_0 can be extended to an elementary homomorphism ψ, if the structure induced by the elements where g_0 is not defined has bounded tree width. We fix values every possible way in which the conditions of Lemma 4.3 can be violated and use HOMOMORPHISM-EXT to check whether there is an extension compatible with this choice. In order to efficiently enumerate all the possible violations of the second condition, the following definition is needed:

Given a relation $R^{\mathbb{B}}$ of arity r, a *bad prefix* is a tuple $(b_1, \ldots, b_s) \in B^s$ with $s \leq r$ such that

(1) there is no tuple $(b_1, \ldots, b_s, b_{s+1}, \ldots, b_r) \in R^{\mathbb{B}}$ for any $b_{s+1}, \ldots, b_r \in B$, and
(2) there is a tuple $(b_1, \ldots, b_{s-1}, c_s, c_{s+1}, \ldots, c_r) \in R^{\mathbb{B}}$ for some $c_t, \ldots, c_r \in B$.

If $(b_1, \ldots, b_r) \notin R^{\mathbb{B}}$, then there is a unique $1 \leq s \leq r$ such that the tuple $(b_1, \ldots, b_s)$ is a bad prefix: there has to be an s such that $(b_1, \ldots, b_s)$ cannot be extended to a tuple of $R^{\mathbb{B}}$, but $(b_1, \ldots, b_{s-1})$ can.

Lemma 4.4. *The relation $R^{\mathbb{B}}$ has at most $|R^{\mathbb{B}}| \cdot (|B| - 1) \cdot r$ bad prefixes, where r is the arity of the relation.*

Lemma 4.5. *Let X be a subset of A_t and let g_0 be a mapping from X to B. For every fixed k, there is a polynomial-time algorithm* ELEMENTARY-EXT(t, X, g_0) *that decides whether g_0 can be extended to an elementary homomorphism from $\mathbb{A}_t$ to B, if the tree width of the structure induced by $A_t - X$ is at most k.*

We enumerate the elementary homomorphisms in a specific order defined by the following precedence relation. Let φ be an elementary homomorphism from $\mathbb{A}_i$ to $\mathbb{B}$ and let ψ be an elementary homomorphism from $\mathbb{A}_j$ to $\mathbb{B}$ for some $j > i$. Homomorphism ψ is the *parent* of φ (φ is a *child* of ψ) if φ restricted to A_{i+1} can be written as $\psi \circ \varphi_j \circ \ldots \circ \varphi_{i+2}$. *Ancestor* and *descendant* relations are defined as the reflexive transitive closure of the parent and child relations, respectively.

Note that an elementary homomorphism from $\mathbb{A}_i$ to $\mathbb{B}$ has exactly one parent for $i < n$ and a homomorphism from $\mathbb{A}_n$ to $\mathbb{B}$ has no parent. Fix an arbitrary ordering of the elements of A. For $0 \leq i \leq n$ and $0 \leq j \leq |A_i \setminus A_{i+1}|$, let $A_{i,j}$ be the union of A_{i+1} and the first j elements of $A_i \setminus A_{i+1}$. Note that $A_{i,0} = A_{i+1}$ and $A_{i,|A_i \setminus A_{i+1}|} = A_i$.

Lemma 4.6. *Let ψ be a mapping from $A_{i,j}$ to $\mathbb{B}$ that can be extended to an elementary homomorphism from $\mathbb{A}_i$ to $\mathbb{B}$. Assume that a sequence of width k endomorphisms is given for $\mathbb{A}$. For every fixed k, there is a polynomial-delay, polynomial-space algorithm* ELEMENTARY-ENUM(i, j, ψ) *that enumerates all the elementary homomorphisms of $\mathbb{A}_i$ that extends ψ and all the descendants of these homomorphisms.*

By calling ELEMENTARY-ENUM$(n, 0, g_0)$ (where g_0 is a trivial mapping from $\emptyset$ to $\mathbb{B}$), we can enumerate all the elementary homomorphisms. By the observation in Lemma 4.2, this means that we can enumerate all the homomorphisms from $\mathbb{A}$ to $\mathbb{B}$.

Theorem 4.7. *For every fixed k, there is a polynomial-delay, polynomial-space algorithm that, given structures $\mathbb{A}$, $\mathbb{B}$, and a sequence of width k endomorphisms of $\mathbb{A}$, enumerates all the homomorphisms from $\mathbb{A}$ to $\mathbb{B}$.*

Theorem 4.7 does not provide a complete description of classes of structures solvable WPD.

Corollary 4.8. *There is a class $\mathcal{A}$ of relational structures such that not all structures from $\mathcal{A}$ have a sequence of width k endomorphisms and $\mathrm{ECSP}(\mathcal{A}, -)$ is solvable WPD.*

Proof. Let $\mathcal{A}$ be the class of structures that are the disjoint union of a loop and a core. Obviously, $\mathrm{SCSP}(\mathcal{A}, -)$ is polynomial time solvable. Therefore, by Lemma 3.4, $\mathrm{ECSP}(\mathcal{A}', -)$ is solvable with polynomial delay. However, it is not hard to see that $\mathcal{A}'$ does not have a sequence of endomorphisms of bounded tree width. ∎

Furthermore, as we will see in the next section it is hard, in general, to find a sequence of bounded width endomorphims. Still, we can find a sequence of endomorphisms for a structure $\mathbb{A}$ if we impose two more restrictions on such a sequence.

A retraction φ of a structure $\mathbb{A}$ is called a *k-retraction* if at most k nodes change their value according to φ. A structure is a *k-core* if the only k-retraction is the identity. A k-core of a structure is any k-core obtained by a sequence of k-retractions.

Lemma 4.9. *All k-cores of a structure $\mathbb{A}$ are isomorphic.*

Lemma 4.9 amounts to say that when searching for a sequence of k-retractions converging to a k-core we can use the greedy approach and include, as the next member of such a sequence, any k-retraction with required properties. With this in hands we now can apply Theorem 4.7.

Theorem 4.10. *Let $k > 0$ be a positive integer and let $\mathcal{C}$ be a class of structures such that the k-core of every structure in $\mathcal{C}$ has tree width at most k. Then, the enumeration problem $\mathrm{ECSP}(\mathcal{C}, -)$ is solvable WPD.*

Corollary 4.11. *If $\mathcal{C}$ is a class of structures of bounded tree width then $\mathrm{ECSP}(\mathcal{C}, -)$ is solvable WPD.*

5. Hardness results

The first result of this section shows that finding a sequence of endomorphisms of bounded width can be difficult even in simplest cases.

Theorem 5.1. *It is NP-complete to decide if a structure has a sequence of 1-width retractions to the core.*

The second result shows that $\mathrm{ECSP}(\mathcal{A}, -)$ can be hard even if every structure in $\mathcal{A}$ has a sequence of width-2 endomorphisms. Note that this result is incomparable with Theorem 5.1, since an enumeration algorithm (in theory) does not necessarily have to compute an sequence of endomorphisms. We need the following lemma:

Lemma 5.2. *If G is a planar graph, then it is possible to find a partition (V_1, V_2) of its vertices in polynomial time such that $G[V_1]$ and $G[V_2]$ have tree width at most 2.*

Proposition 5.3. *There is a class $\mathcal{A}$ of relational structures such that every structure from $\mathcal{A}$ has a sequence of width 2 endomorphisms to the core, and such that the problem $\mathrm{ECSP}(\mathcal{A}, -)$ is not solvable WPD, unless $P = NP$.*

Proof. Let $\mathcal{A}$ be a class of graphs built in the following way. Take a 3-colorable planar graph G and its partition (V_1, V_2) according to Lemma 5.2. Using colorings we can ensure that G is a core. Then we take a disjoint union of this graph with a triangle T having all the colors and a copy G_1 of $G[V_1]$. Let $\mathbb{A}$ denote the resulting structure.

CLAIM 1. $\mathbb{A}$ has a sequence of width-2 endomorphisms.

Let ψ be a 3-coloring of G that is a homomorphism into the triangle, and ψ' the bijective mapping from G_1 to $G[V_1]$. Then φ_1 is defined to act as ψ on G, as ψ' on G'_1 and identically on T. Endomorphism φ_2 is just the 3-coloring of $G \cup G_1$ induced by ψ. The images of φ_1 and φ_2 are $T \cup G[V_1]$ and T, respectively, so all the conditions on a sequence of width-2 homomorphisms are easily checkable.

CLAIM 2. The PLANAR GRAPH 3-COLORING PROBLEM is Turing reducible to $\mathrm{ECSP}(\mathcal{A}, -)$.

Given a planar graph G we find its partition (V_1, V_2) and create a structure $\mathbb{A}$, as described above. Then we apply an algorithm that enumerates solutions to $\mathrm{ECSP}(\mathcal{A}, -)$ We may assume that such an algorithm stops with some time bound regardless whether G is 3-colorable or not. If the algorithm succeeds we can now produce a 3-coloring of G. ∎

6. Conjunctive queries

When making a query to a database one usually needs to obtain values of only those variables (attributes) (s)he is interested in. In terms of homomorphisms this can be translated as follows: For relational structures $\mathbb{A}$, $\mathbb{B}$, and a subset $Y \subseteq A$, we aim to list those mappings from Y to B which can be extended to a full homomorphism from $\mathbb{A}$ to $\mathbb{B}$. In other words, we would like to enumerate all the mappings from Y to B that arise as the restriction of some homomorphism from $\mathbb{A}$ to $\mathbb{B}$. Clearly, this problem significantly differs from the regular enumeration problem. A mapping from Y to B can be extendible to a homomorphism in many ways, possibly superpolynomially many, and an enumeration algorithm would list all of them. In the worst case scenario it would list them before turning to the next partial mapping. If this happens it may destroy polynomiality of the delay between outputting consecutive solutions.

In this section we treat the CONJUNCTIVE QUERY EVALUATION PROBLEM as follows.

$\mathrm{CQE}(\mathcal{A},\mathcal{B})$
Instance: $\mathbb{A}\in\mathcal{A}$, $\mathbb{B}\in\mathcal{B}$, $Y\subseteq A$
Problem: Output all partial mappings from Y to B extendible to a homomorphism from $\mathbb{A}$ to $\mathbb{B}$.

We present two results, first one of them shows that the problem $\mathrm{CQE}(\mathcal{A},-)$ is WPD when $\mathcal{A}$ is a class of structures of bounded tree width, the second one claims that, modulo some complexity assumptions, in contrast to enumeration problems this cannot be generalized to structures with k-cores of bounded tree width for $k\geq 2$.

Theorem 6.1. *If $\mathcal{A}$ is a class of structures of bounded width then $\mathrm{CQE}(\mathcal{A},-)$ is solvable WPD.*

Proof. We use Lemma 4.1 to show that algorithm CQE-BOUNDED-WIDTH of Figure 1 does the job. Indeed, this algorithms backtracks only if outputs a solution. ∎

Theorem 6.1 does not generalize to classes of structures whose k-cores have bounded width.

Example 6.2. Recall that the MULTICOLORED CLIQUE problem (cf. [15]) is formulated as follows: Given a number k and a vertex k-colored graph, decide if the graph contains a k-clique all vertices of which are colored different colors. This problem is $W[1]$-complete, i.e., has no time $f(k)n^c$ algorithm for any function f and constant c, unless FPT$=W[1]$. We reduce this problem to $\mathrm{CQE}(\mathcal{A},-)$ where $\mathcal{A}$ is the class of structures whose 2-cores are 2-element described below.

Let us consider relational structures with two binary and two unary relations. This structure can be thought of as a graph whose vertices and edges have one of the two colors, say, red and blue, accordingly to which of the two binary/unary relations they belong to. Let $\mathbb{A}_k$ be the relational structure with universe $\{a_1,\dots,a_k,y_1,\dots,y_k\}$, where $a_1,\dots,a_k$ are red while $y_1,\dots,y_k$ are blue. Then $\{a_1,\dots,a_k\}$ induces a red clique, that is every a_i,a_j (i,j are not necessarily different) are connected with a red edge, and each y_i is connected to a_i with a blue edge. It is not hard to see that every pair of a red and blue vertices induces a 2-core of this structure. Set $\mathcal{A}=\{\mathbb{A}_k\mid k\in\mathbb{N}\}$.

The reduction of the MULTICOLORED CLIQUE problem to $\mathrm{CQE}(\mathcal{A},-)$ goes as follows. Given a k-colored graph $G=(V,E)$ whose coloring induces a partition of V into classes $B_1,\dots,B_k$. Then we define structures $\mathbb{A},\mathbb{B}$ and a set $Y\subseteq A$. We set $\mathbb{A}=\mathbb{A}_k$, $Y=\{y_1,\dots,y_k\}$. Then let $B=V\cup\{b_1,\dots,b_k\}$, the elements of V are colored red and the induced substructure $\mathbb{B}[V]$ is the

Figure 1: Algorithm CQE-BOUNDED-WIDTH

Input: Relational structures $\mathbb{A}$, $\mathbb{B}$, and $Y=\{Y_1,\dots,Y_\ell\}\subseteq A$

Output: A list of mappings $\varphi\colon Y\to B$ extendible to a homomorphism from $\mathbb{A}$ to $\mathbb{B}$

Step 1 **set** $m=0$, $\varphi=\emptyset$, $S_i=B$, $i\in[m]$, complete:=**false**
Step 2 **while** not complete **do**
Step 2.1 **if** $m<\ell$ **then do**
Step 2.1.1 **search** S_{m+1} until a $b\in S_{m+1}$ is found such that there exists a homomorphism extending $\varphi\cup\{y_{m+1}\to b\}$ and **remove** all members of S_{m+1} preceding b inclusive
Step 2.1.2 **if** such a b exists **then set** $\varphi:=\varphi\cup\{y_{m+1}\to b\}$, $m:=m+1$
Step 2.1.3 **else**
Step 2.1.3.1 **if** $m\neq 0$ **then set** $\varphi=\varphi_{|\{y_1,\dots,y_{m-1}\}}$ and $S_{m+1}:=B$, $m:=m-1$
Step 2.1.3.2 **else set** complete:=**true**
Step 2.2 **else then do**
Step 2.2.1 **output** φ
Step 2.2.2 **set** $\varphi:=\varphi_{|\{y_1,\dots,y_{m-1}\}\}}$, $m:=\ell-1$
endwhile

graph G (without coloring) whose edges are colored also red. Finally, $b_1, \dots, b_k$ are made blue and each b_i is connected with a blue edge with every vertex from B_i.

It is not hard to see that any homomorphism maps $\{a_1, \dots, a_k\}$ to V and Y to $\{b_1, \dots, b_k\}$, and that the number of homomorphisms that do not agree on Y does not exceed k^k. Moreover, G contains a k-colored clique if and only if there is a homomorphism from $\mathbb{A}$ to $\mathbb{B}$ that maps Y onto $\{b_1, \dots, b_k\}$. If there existed an algorithm solving $\mathrm{CQE}(\mathcal{A}, -)$ WPD, say, time needed to compute the first and every consequent solution is bounded by a polynomial $p(n)$, then time needed to list all solutions is at most $k^k p(n)$. This means that MULTICOLORED CLIQUE is FPT, a contradiction.

References

[1] A. Atserias, A. Bulatov, and V. Dalmau. On the power of *k*-consistency. In *ICALP*, pages 279–290, 2007.

[2] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *FOCS*, 739–748, 2008.

[3] P. Austrin. Towards sharp inapproximability for any 2-csp. In *FOCS*, pages 307–317, 2007.

[4] L. Barto, M. Kozik, and T. Niven. Graphs, polymorphisms and the complexity of homomorphism problems. In *STOC*, pages 789–796, 2008.

[5] M. Bodirsky and J. Kára. The complexity of temporal constraint satisfaction problems. In *STOC*, pages 29–38, 2008.

[6] A. Bulatov. The complexity of the counting constraint satisfaction problem. In *ICALP*, pages 646–661, 2008.

[7] A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *FOCS*, pages 562–571, 2003.

[8] A. Bulatov and M. Grohe. The complexity of partition functions. *Theoret. Comput. Sci.*, 348:148–186, 2005.

[9] A.A. Bulatov. Tractable conservative constraint satisfaction problems. In *LICS*, pages 321–330, 2003.

[10] A.A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM*, 53(1):66–120, 2006.

[11] A.A. Bulatov, P. Jeavons, and A. A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.

[12] H. Chen and V. Dalmau. Beyond Hypertree Width: Decomposition Methods Without Decompositions. In *CP*, pages 167–181, 2005.

[13] M.E. Dyer, L.A. Goldberg, and M. Paterson. On counting homomorphisms to directed acyclic graphs. *J. ACM*, 54(6), 2007.

[14] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM Journal of Computing*, 28:57–104, 1998.

[15] M.R. Fellows, D. Hermelin, F. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems, 2008. Manuscript.

[16] G. Gottlob, L. Leone, and F. Scarcello. Hypertree decomposition and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[17] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), 2007.

[18] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[19] P. Hell and J. Nešetril. The core of graph. *Discrete Mathematics*, 109(1-3):117–126, 1992.

[20] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Ser.B*, 48:92–110, 1990.

[21] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

[22] P. Jonsson, M. Klasson, and A.A. Krokhin. The approximability of three-valued MAX CSP. *SIAM J. Comput.*, 35(6):1329–1349, 2006.

[23] P. Jonsson and A.A. Krokhin. Maximum h-colourable subdigraphs and constraint optimization with arbitrary weights. *J. Comput. Syst. Sci.*, 73(5):691–702, 2007.

[24] P. Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *STOC*, pages 245–254, 2008.

[25] H. Schnoor and I. Schnoor. Enumerating all solutions for constraint satisfaction problems. In *STACS*, pages 694–705, 2007.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 243–254
www.stacs-conf.org

HARDNESS AND ALGORITHMS FOR RAINBOW CONNECTIVITY

SOURAV CHAKRABORTY [1] AND ELDAR FISCHER [1] AND ARIE MATSLIAH [2] AND RAPHAEL YUSTER [3]

[1] Department of Computer Science, Technion, Haifa 32000, Israel.
E-mail address, S. Chakraborty: sourav@cs.technion.ac.il
E-mail address, E. Fischer: eldar@cs.technion.ac.il

[2] Centrum Wiskunde & Informatica (CWI), Amsterdam, Netherlands.
E-mail address, A. Matsliah: ariem@cwi.nl

[3] Department of Mathematics, University of Haifa, Haifa 31905, Israel.
E-mail address, R. Yuster: raphy@math.haifa.ac.il

ABSTRACT. An edge-colored graph G is *rainbow connected* if any two vertices are connected by a path whose edges have distinct colors. The *rainbow connectivity* of a connected graph G, denoted $rc(G)$, is the smallest number of colors that are needed in order to make G rainbow connected. In addition to being a natural combinatorial problem, the rainbow connectivity problem is motivated by applications in cellular networks. In this paper we give the first proof that computing $rc(G)$ is NP-Hard. In fact, we prove that it is already NP-Complete to decide if $rc(G) = 2$, and also that it is NP-Complete to decide whether a given edge-colored (with an unbounded number of colors) graph is rainbow connected. On the positive side, we prove that for every $\epsilon > 0$, a connected graph with minimum degree at least ϵn has bounded rainbow connectivity, where the bound depends only on ϵ, and the corresponding coloring can be constructed in polynomial time. Additional non-trivial upper bounds, as well as open problems and conjectures are also presented.

1. Introduction

Connectivity is perhaps the most fundamental graph-theoretic property, both in the combinatorial sense and the algorithmic sense. There are many ways to strengthen the connectivity property, such as requiring hamiltonicity, k-connectivity, imposing bounds on the diameter, requiring the existence of edge-disjoint spanning trees, and so on.

An interesting way to quantitavely strengthen the connectivity requirement was recently introduced by Chartrand et al. in [5]. An edge-colored graph G is *rainbow connected* if any two vertices are connected by a path whose edges have distinct colors. Clearly, if a graph is rainbow connected, then it is also connected. Conversely, any connected graph has a trivial edge coloring that makes it rainbow connected; just color each edge with a distinct color. Thus, one can properly define the

The work was done when Sourav Chakraborty was a Phd student at University of Chicago and Arie Matsliah was a Phd student at Technion, Israel.

rainbow connectivity of a connected graph G, denoted $rc(G)$, as the smallest number of colors that are needed in order to make G rainbow connected. An easy observation is that if G is connected and has n vertices then $rc(G) \leq n-1$, since one may color the edges of a given spanning tree with distinct colors. We note also the trivial fact that $rc(G) = 1$ if and only if G is a clique, the (almost) trivial fact that $rc(G) = n-1$ if and only if G is a tree, and the easy observation that a cycle with $k > 3$ vertices has rainbow connectivity $\lceil k/2 \rceil$. Also notice that, clearly, $rc(G) \geq \mathrm{diam}(G)$ where $\mathrm{diam}(G)$ denotes the diameter of G.

Chartrand et al. computed the rainbow connectivity of several graph classes including complete multipartite graphs [5]. Caro et al. [6] considered the extremal graph-theoretic aspects of rainbow connectivity. They proved that if G is a connected graph with n vertices and with minimum degree 3 then $rc(G) < 5n/6$, and if the minimum degree is δ then $rc(G) \leq \frac{\ln \delta}{\delta} n(1+f(\delta))$ where $f(\delta)$ tends to zero as δ increases. They also determine the threshold function for a random graph $G(n, p(n))$ to have $rc(G) = 2$. In their paper, they conjecture that computing $rc(G)$ is an NP-Hard problem, as well as conjecture that even deciding whether a graph has $rc(G) = 2$ in NP-Complete.

In this paper we address the computational aspects of rainbow connectivity. Our first set of results solve, and extend, the complexity conjectures from [6]. Indeed, it turns out that deciding whether $rc(G) = 2$ is an NP-Complete problem. Our proof is by a series of reductions, where on the way it is shown that 2-rainbow-colorability is computationally equivalent to the seemingly harder question of deciding the existence of a 2-edge-coloring that is required to rainbow-connect only vertex pairs from a prescribed set.

Theorem 1.1. *Given a graph G, deciding if $rc(G) = 2$ is NP-Complete. In particular, computing $rc(G)$ is NP-Hard.*

Suppose we are given an edge coloring of the graph. Is it then easier to verify whether the colored graph is rainbow connected? Clearly, if the number of colors in constant then this problem becomes easy. However, if the coloring is arbitrary, the problem becomes NP-Complete:

Theorem 1.2. *The following problem is NP-Complete: Given an edge-colored graph G, check whether the given coloring makes G rainbow connected.*

For the proof of Theorem 1.2, we first show that the $s-t$ version of the problem is NP-Complete. That is, given two vertices s and t of an edge-colored graph, decide whether there is a rainbow path connecting them.

We now turn to positive algorithmic results. Our main positive result is that connected n-vertex graphs with minimum degree $\Theta(n)$ have *bounded* rainbow connectivity. More formally, we prove:

Theorem 1.3. *For every $\epsilon > 0$ there is a constant $C = C(\epsilon)$ such that if G is a connected graph with n vertices and minimum degree at least ϵn, then $rc(G) \leq C$. Furthermore, there is a polynomial time algorithm that constructs a corresponding coloring for a fixed ϵ.*

The proof of Theorem 1.3 is based upon a modified degree-form version of Szemerédi's Regularity Lemma that we prove and that may be useful in other applications. From our algorithm it is also not hard to find a probabilistic polynomial time algorithm for finding this coloring with high probability (using on the way the algorithmic version of the Regularity Lemma from [1] or [7]).

We note that connected graphs with minimum degree ϵn have bounded diameter, but the latter property by itself does *not* guarantee bounded rainbow connectivity. As an extreme example, a star

with n vertices has diameter 2 but its rainbow connectivity is $n-1$. The following theorem asserts however that having diameter 2 and only logarithmic minimum degree suffices to guarantee rainbow connectivity 3.

Theorem 1.4. *If G is an n-vertex graph with diameter 2 and minimum degree at least $8\log n$ then $rc(G) \leq 3$. Furthermore, such a coloring is given with high probability by a uniformly random 3-edge-coloring of the graph G, and can also be found by a polynomial time deterministic algorithm.*

Since a graph with minimum degree $n/2$ is connected and has diameter 2, we have as an immediate corollary:

Corollary 1.5. *If G is an n-vertex graph with minimum degree at least $n/2$ then $rc(G) \leq 3$.*

The rest of this paper is organized as follows. The next section contains the hardness results, including the proofs of Theorem 1.1 and Theorem 1.2. Section 3 contains the proof of Theorem 1.3 and the proof of Theorem 1.4. At the end of the proof of each of the above theorems we explain how the algorithm can be derived – this mostly consists of using the conditional expectation method to derandomize the probabilistic parts of the proofs. The final Section 4 contains some open problems and conjectures. Due to space limitations, several proofs have been omitted from this write-up.

2. Hardness results

We first give an outline of our proof of Theorem 1.1. We begin by showing the computational equivalence of the problem of rainbow connectivity 2, that asks for a red-blue edge coloring in which *all* vertex pairs have a rainbow path connecting them, to the problem of *subset rainbow connectivity* 2, asking for a red-blue coloring in which every pair of vertices in a *given subset* of pairs has a rainbow path connecting them. This is proved in Lemma 2.1 below.

In the second step, we reduce the problem of *extending to rainbow connectivity* 2, asking whether a given partial red-blue coloring can be completed to a obtain a rainbow connected graph, to the subset rainbow connectivity 2 problem. This is proved in Lemma 2.2 below.

Finally, the proof of Theorem 1.1 is completed by reducing 3-SAT to the problem of *extending to rainbow connectivity* 2.

Lemma 2.1. *The following problems are polynomially equivalent:*

(1) *Given a graph G decide whether $rc(G) = 2$.*
(2) *Given a graph G and a set of pairs $P \subseteq V(G) \times V(G)$, decide whether there is an edge coloring of G with 2 colors such that all pairs $(u, v) \in P$ are rainbow connected.*

Lemma 2.2. *The first problem defined below is polynomially reducible to the second one:*

(1) *Given a graph $G = (V, E)$ and a partial 2-edge-coloring $\hat{\chi} : \hat{E} \to \{0, 1\}$ for $\hat{E} \subset E$, decide whether $\hat{\chi}$ can be extended to a complete 2 edge-coloring $\chi : E \to \{0, 1\}$ that makes G rainbow connected.*
(2) *Given a graph G and a set of pairs $P \subseteq V(G) \times V(G)$ decide whether there is an edge coloring of G with 2 colors such that all pairs $(u, v) \in P$ are rainbow connected.*

We are unable to present the proofs of Lemma 2.1 and Lemma 2.2 due to space limitations.

Proof of Theorem 1.1 We show that Problem 1 of Lemma 2.2 is NP-hard, and then deduce that 2-rainbow-colorability is NP-Complete by applying Lemma 2.1 and Lemma 2.2 while observing that it clearly belongs to NP.

We reduce 3-SAT to Problem 1 of Lemma 2.2. Given a 3CNF formula $\phi = \bigwedge_{i=1}^{m} c_i$ over variables $x_1, x_2, \ldots, x_n$, we construct a graph G_ϕ and a partial 2-edge coloring $\chi' : E(G_\phi) \to \{0,1\}$ such that there is an extension χ of χ' that makes G_ϕ rainbow connected if and only if ϕ is satisfiable.

We define G_ϕ as follows:

$$V(G_\phi) = \{c_i : i \in [m]\} \cup \{x_i : i \in [n]\} \cup \{a\}$$

$$E(G_\phi) = \Big\{\{c_i, x_j\} : x_j \in c_i \text{ in } \phi\Big\} \cup \Big\{\{x_i, a\} : i \in [n]\Big\} \cup \Big\{\{c_i, c_j\} : i, j \in [m]\Big\} \cup \Big\{\{x_i, x_j\} : i, j \in [n]\Big\}$$

and we define the partial coloring χ' as follows:

$$\forall_{i,j \in [m]} \chi'(\{c_i, c_j\}) = 0$$

$$\forall_{i,j \in [n]} \chi'(\{x_i, x_j\}) = 0$$

$$\forall_{\{x_i, c_j\} \in E(G_\phi)} \chi'(\{x_i, c_j\}) = 0 \text{ if } x_i \text{ is positive in } c_j,\ 1 \text{ otherwise}$$

while all the edges in $\Big\{\{x_i, a\} : i \in [n]\Big\}$ (and only they) are left uncolored.

Assuming without loss of generality that all variables in ϕ appear both as positive and as negative, one can verify that a 2-rainbow-coloring of the uncolored edges corresponds to a satisfying assignment of ϕ and vice versa. ∎

The proof of Theorem 1.2 is based upon the proof of the following theorem.

Theorem 2.3. *The following problem is NP-complete: Given an edge colored graph G and two vertices s, t of G, decide whether there is a rainbow path connecting s and t.*

Proof. Clearly the problem is in NP. We prove that it is NP-Complete by reducing 3-SAT to it. Given a 3CNF formula $\phi = \bigwedge_{i=1}^{m} c_i$ over variables $x_1, x_2, \ldots, x_n$, we construct a graph G_ϕ with two special vertices s, t and a coloring $\chi : E(G_\phi) \to [|E(G_\phi)|]$ such that there is a rainbow path connecting s and t in G_ϕ if and only if ϕ is satisfiable.

We start by constructing an auxiliary graph G' from ϕ. The graph G' has $3m + 2$ vertices, that are partitioned into $m + 2$ layers $V_0, V_1, \ldots, V_m, V_{m+1}$, where $V_0 = \{s\}$, $V_{m+1} = \{t\}$ and for each $i \in [m]$, the layer V_i contains the three vertices corresponding to the literals of c_i (a clause in ϕ). The edges of G' connect between all pairs of vertices residing in consecutive layers. Formally,

$$E(G') = \Big\{\{u, v\} : \exists i \in [m+1] \text{ s.t. } u \in V_{i-1} \text{ and } v \in V_i\Big\}.$$

Intuitively, in our final colored graph G_ϕ, every rainbow path from s to t will define a satisfying assignment of ϕ in a way that for every $i \in [m]$, if the rainbow path contains a vertex $v \in V_i$ then the literal of c_i that corresponds to v is satisfied, and hence c_i is satisfied. Since any path from s to t must contain at least one vertex from every layer V_i, this will yield a satisfying assignment for the whole formula ϕ. But we need to make sure that there are no contradictions in this assignment,

that is, no opposite literals are satisfied together. For this we modify G' by replacing each literal-vertex with a gadget, and we define an edge coloring for which rainbow paths yield only consistent assignments.

For every variable x_j, $j \in [n]$, let $v_{j_1}, v_{j_2}, \ldots, v_{j_k}$ be the vertices of G' corresponding to the positive literal x_j, and let $\overline{v}_{j_1}, \overline{v}_{j_2}, \ldots, \overline{v}_{j_\ell}$ be the vertices corresponding to the negative literal $\overline{x}_j$. We can assume without loss of generality that both $k \geq 1$ and $\ell \geq 1$, since otherwise the formula ϕ can be simplified. For every such variable x_j we also introduce $k \times \ell$ distinct colors $\alpha^j_{1,1}, \ldots, \alpha^j_{k,\ell}$. Next, we transform the auxiliary graph G' into the final graph G_ϕ.

For every $a \in [k]$ we replace the vertex v_{j_a} that resides in layer (say) V_i with $\ell+1$ new vertices $v_1, v_2, \ldots, v_{\ell+1}$ that form a path in that order. We also connect all vertices in V_{i-1} to v_1 and connect all vertices in V_{i+1} to $v_{\ell+1}$. For every $b \in [\ell]$, we color the edge $\{v_b, v_{b+1}\}$ in the new path with the color $\alpha^j_{a,b}$. Similarly, for every $b \in [\ell]$ we replace the vertex $\overline{v}_{j_b}$ from layer (say) $V_{i'}$ with $k+1$ new vertices $\overline{v}_1, \overline{v}_2, \ldots, \overline{v}_{k+1}$ that form a path, and connect all vertices in $V_{i'-1}$ to $\overline{v}_1$ and all vertices in $V_{i'+1}$ to $\overline{v}_{k+1}$. For every $a \in [k]$, we color the edge $\{v_a, v_{a+1}\}$ with $\alpha^j_{a,b}$. All other edges of G_ϕ (which were the original edges of G') are colored with fresh distinct colors.

Clearly, any path from s to t in G_ϕ must contain at least one of the newly built paths in each layer. On the other hand, it is not hard to verify that any two paths of opposite literals of the same variable have edges sharing the same color. ∎

Proof of Theorem 1.2. We reduce from the problem in Theorem 2.3. Given an edge colored graph $G = (V, E)$ with two special vertices s and t, we construct a graph $G' = (V', E')$ and define a coloring $\chi' : E' \to [|E'|]$ of its edges such that s and t are rainbow connected in G if and only if the coloring of G' makes G' rainbow connected.

Let $V = \{v_1 = s, v_2, \ldots, v_n = t\}$ be the vertices of the original graph G. We set

$$V' = V \cup \{s', t', b\} \cup \{s^1, v_2^1, v_2^2, \ldots, v_{n-1}^1, v_{n-1}^2, t^2\}$$

and

$$E' = E \cup \Big\{\{s', s\}, \{t', t\}, \{s, s^1\}, \{t, t^2\}\Big\} \cup \Big\{\{b, v_i\} : i \in [n]\Big\} \cup$$
$$\cup \Big\{\{v_i, v_i^j\} : i \in [n],\ j \in \{1, 2\}\Big\} \cup \Big\{\{v_i^a, v_j^b : i, j \in [n],\ a, b \in \{1, 2\}\Big\}.$$

The coloring χ' is defined as follows:

- all edges $e \in E$ retain the original color, that is $\chi'(e) = \chi(e)$;
- the edges $\{t, t'\}, \{s, b\}$ and $\Big\{\{v_i, v_i^1\} : i \in [n-1]\Big\}$ are colored with a special color c_1;
- the edges $\{s, s'\}, \{t, b\}$ and $\Big\{\{v_i, v_i^2\} : i \in [2, n]\Big\}$ are colored with a special color c_2;
- the edges in $\Big\{\{v_i, b\} : i \in [2, n-1]\Big\}$ are colored with a special color c_3;
- the edges in $\Big\{\{v_i^a, v_j^b\} : i, j \in [n],\ a, b \in \{1, 2\}\Big\}$ are colored with a special color c_4.

One can verify that χ' makes G' rainbow connected if and only if there was a rainbow path from s to t in G. ∎

3. Upper bounds and algorithms

The proof of our main Theorem 1.3 is based upon a modified degree-form version of Szemerédi's Regularity Lemma, that we prove here and that may be useful in other applications. We begin by introducing the Regularity Lemma and the already known degree-form version of it.

3.1. Regularity Lemma

The Regularity Lemma of Szemerédi [9] is one of the most important results in graph theory and combinatorics, as it guarantees that every graph has an ϵ-approximation of constant descriptive size, namely a size that depends only on ϵ and not on the size of the graph. This approximation "breaks" the graph into a constant number of pseudo-random bipartite graphs. This is very useful in many applications since dealing with random-like graphs is much easier than dealing with arbitrary graphs. In particular, as we shall see, the Regularity Lemma allows us to prove that graphs with linear minimum degree have bounded rainbow connectivity.

We first state the lemma. For two nonempty disjoint vertex sets A and B of a graph G, we define $E(A,B)$ to be the set of edges of G between A and B. The *edge density* of the pair is defined by $d(A,B) = |E(A,B)|/(|A||B|)$.

Definition 3.1 (ϵ-regular pair)**.** A pair (A,B) is *ϵ-regular* if for every $A' \subseteq A$ and $B' \subseteq B$ satisfying $|A'| \geq \epsilon|A|$ and $|B'| \geq \epsilon|B|$, we have $|d(A',B') - d(A,B)| \leq \epsilon$.

An ϵ-regular pair can be thought of as a pseudo-random bipartite graph in the sense that it behaves almost as we would expect from a random bipartite graph of the same density. Intuitively, in a random bipartite graph with edge density d, all large enough sub-pairs should have similar densities.

A partition $V_1, \ldots, V_k$ of the vertex set of a graph is called an *equipartition* if $|V_i|$ and $|V_j|$ differ by no more than 1 for all $1 \leq i < j \leq k$ (so in particular every V_i has one of two possible sizes). The *order* of an equipartition denotes the number of partition classes (k above). An equipartition $V_1, \ldots, V_k$ of the vertex set of a graph is called *ϵ-regular* if all but at most $\epsilon\binom{k}{2}$ of the pairs (V_i, V_j) are ϵ-regular. Szemerédi's Regularity Lemma can be formulated as follows.

Lemma 3.2 (Regularity Lemma [9])**.** *For every $\epsilon > 0$ and positive integer K, there exists $N = N_{3.2}(\epsilon, K)$, such that any graph with $n \geq N$ vertices has an ϵ-regular equipartition of order k, where $K \leq k \leq N$.*

As mentioned earlier, the following variation of the lemma comes useful in our context.

Lemma 3.3 (Regularity Lemma - degree form [8])**.** *For every $\epsilon > 0$ and positive integer K there is $N = N_{3.3}(\epsilon, K)$ such that given any graph $G = (V,E)$ with $n > N$ vertices, there is a partition of the vertex-set V into $k+1$ sets $V_0', V_1', \ldots, V_k'$, and there is a subgraph G' of G with the following properties:*

(1) *$K \leq k \leq N$,*
(2) *$s \triangleq |V_0'| \leq \epsilon^5 n$ and all other components V_i', $i \in [k]$ are of size $\ell \triangleq \frac{n-s}{k}$,*
(3) *for all $i \in [k]$, V_i' induces an independent set in G',*
(4) *for all $i, j \in [k]$, the pair (V_i', V_j') is ϵ^5-regular in G', with density either 0 or at least $\frac{\epsilon}{4}$,*

(5) *for all* $v \in V$, $\deg_{G'}(v) > \deg_G(v) - \frac{\epsilon}{3}n$.

This form of the lemma (see e.g. [8]) can be obtained by applying the original Regularity Lemma (with a smaller value of ϵ), and then "cleaning" the resulting partition. Namely, adding to the exceptional set V_0' all components V_i incident to many irregular pairs, deleting all edges between any other pairs of clusters that either do not form an ϵ-regular pair or they do but with density less than ϵ, and finally adding to V_0 also vertices whose degree decreased too much by this deletion of edges.

3.2. A modified degree form version of the Regularity Lemma

In order to prove that graphs with linear minimum degree have bounded rainbow connectivity number, we need a special version of the Regularity Lemma, which is stated next.

Lemma 3.4 (Regularity Lemma - new version)**.** *For every $\epsilon > 0$ and positive integer K there is $N = N_{3.4}(\epsilon, K)$ so that the following holds: If $G = (V, E)$ is a graph with $n > N$ vertices and minimum degree at least ϵn then there is a subgraph G'' of G, and a partition of V into $V_1'', \ldots, V_k''$ with the following properties:*

(1) $K \leq k \leq N$,
(2) *for all* $i \in [k]$, $(1-\epsilon)\frac{n}{k} \leq |V_i''| \leq (1+\epsilon^3)\frac{n}{k}$,
(3) *for all* $i \in [k]$, V_i'' *induces an independent set in* G'',
(4) *for all* $i, j \in [k]$, (V_i'', V_j'') *is an* ϵ^3*-regular pair in* G''*, with density either* 0 *or at least* $\frac{\epsilon}{16}$,
(5) *for all* $i \in [k]$ *and every* $v \in V_i''$ *there is at least one other class* V_j'' *so that the number of neighbors of* v *in* G'' *belonging to* V_j'' *is at least* $\frac{\epsilon}{2}|V_j''|$.

We also note that the above a partition as guaranteed by our modified version of the Regularity Lemma can be found in polynomial time for a fixed ϵ (with somewhat worse constants), by using the exact same methods that were used in [1] for constructing an algorithmic version of the original Regularity Lemma. We are unable to give the complete proof of Lemma 3.4 due to space limitations.

3.3. Proof of Theorem 1.3

In this section we use our version of the Regularity Lemma to prove Theorem 1.3. First we need some definitions. Given a graph $G = (V, E)$ and two subsets $V_1, V_2 \subseteq V$, let $E(V_1, V_2)$ denote the set of edges having one endpoint in V_1 and another endpoint in V_2. Given a vertex v, let $\Gamma(v)$ denote the set of v's neighbors, and for $W \subseteq V$, let $\Gamma_W(v)$ denote the set $W \cap \Gamma(v)$.

For an edge coloring $\chi : E \to \mathcal{C}$, let π_χ denote the corresponding partition of E into (at most) $|\mathcal{C}|$ components. For two edge colorings χ and χ', we say that χ' is a *refinement* of χ if $\pi_{\chi'}$ is a refinement of π_χ, which is equivalent to saying that $\chi'(e_1) = \chi'(e_2)$ always implies $\chi(e_1) = \chi(e_2)$.

Observation 3.5. Let χ and χ' be two edge-colorings of a graph G, such that χ' is a refinement of χ. For any path P in G, if P is a rainbow path under χ, then P is a rainbow path under χ'. In particular, if χ makes G rainbow connected, then so does χ'. ∎

We define a set of eight distinct colors $\mathcal{C} = \{a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4\}$. Given a coloring $\chi : E \to \mathcal{C}$ we say that $u, v \in V$ are *a-rainbow connected* if there is a rainbow path from u to v using only the colors a_1, a_2, a_3, a_4. We similarly define *b-rainbow connected* pairs. The following is a central lemma in the proof of Theorem 1.3. The proof of Lemma 3.6 is given in Section 3.4.

Lemma 3.6. *For any $\epsilon > 0$, there is $N = N_{3.6}(\epsilon)$ such that any connected graph $G = (V, E)$ with $n > N$ vertices and minimum degree at least ϵn satisfies the following. There is a partition Π of V into $k \leq N$ components $V_1, V_2, \ldots, V_k$, and a coloring $\chi : E \to \mathcal{C}$ such that for every $i \in [k]$ and every $u, v \in V_i$, the pair u, v is both a-rainbow connected and b-rainbow connected under χ.*

Using Lemma 3.6 we derive the proof of Theorem 1.3. For a given $\epsilon > 0$, set $N = N_{3.6}(\epsilon)$ and set $C = \frac{3}{\epsilon}N + 8$. Clearly, any connected graph $G = (V, E)$ with $n \leq C$ vertices satisfies $rc(G) \leq C$. So we assume that $n > C \geq N$, and let $\Pi = V_1, \ldots, V_k$ be the partition of V from Lemma 3.6, while we know that $k \leq N$.

First observe that since the minimal degree of G is ϵn, the diameter of G is bounded by $3/\epsilon$. This can be verified by e.g. by taking an arbitrary vertex $r \in V$ and executing a BFS algorithm from it. Let $L_1, \ldots, L_t$ be the layers of vertices in this execution, where L_i are all vertices at distance i from r. Observe that since the minimal degree is at least ϵn, the total number of vertices in every three consecutive layers must be at least ϵn, thus $t \leq 3/\epsilon$. Since the same claim holds for any $r \in V$, this implies that $\mathrm{diam}(G) \leq t \leq 3/\epsilon$.

Now let $T = (V_T, E_T)$ be a connected subtree of G on at most $k \cdot \mathrm{diam}(G) \leq \frac{3}{\epsilon}N$ vertices such that for every $i \in [k]$, $V_T \cap V_i \neq \emptyset$. Such a subtree must exist in G since as observed earlier, $\mathrm{diam}(G) \leq 3/\epsilon$. Let $\chi : E \to \mathcal{C}$ be the coloring from Lemma 3.6, and let $\mathcal{H} = \{h_1, h_2, \ldots, h_{|E_T|}\}$ be a set of $|E_T| \leq \frac{3}{\epsilon}N$ fresh colors. We refine χ by recoloring every $e_i \in E(T)$ with color $h_i \in \mathcal{H}$. Let $\chi' : E \to \Big(\mathcal{C} \cup \mathcal{H}\Big)$ be the resulting coloring of G. The following lemma completes the proof of Theorem 1.3.

Lemma 3.7. *The coloring χ' makes G rainbow connected. Consequently, $rc(G) \leq |E_T| + 8 \leq C$.*

Proof. Let $u, v \in V$ be any pair of G's vertices. If u and v reside in the same component V_i of the partition Π, then (by Lemma 3.6) they are connected by a path P of length at most four, which is a rainbow path under the the original coloring χ. Since χ' is a refinement of χ, the path P remains a rainbow path under χ' as well (see Observation 3.5).

Otherwise, let $u \in V_i$ and $v \in V_j$ for $i \neq j$. Let t_i and t_j be vertices of the subtree T, residing in V_i and V_j respectively. By definition of χ', there is a rainbow path from t_i to t_j using colors from $\mathcal{H}$. Let P_t denote this path. In addition, by Lemma 3.6 we know that for the original coloring χ, there is a rainbow path P_a from u to t_i using colors $a_1, \ldots, a_4$ and there is a rainbow path P_b from v to t_j using colors $b_1, \ldots, b_4$. Based on the fact that χ' is a refinement of χ, it is now easy to verify that P_t, P_a and P_b can be combined to form a rainbow path from u to v under χ'. ∎

This concludes the proof of Theorem 1.3, apart from the existence of a polynomial time algorithm for finding this coloring. We note that all arguments above apart from Lemma 3.6 admit polynomial algorithms for finding the corresponding structures. The algorithm for Lemma 3.6 will be given with its proof.

3.4. Proof of Lemma 3.6

First we state another auxiliary lemma, which is proved in the next section.

Lemma 3.8. *For every $\epsilon > 0$ there exists $N = N_{3.8}(\epsilon)$ such that any graph $G = (V, E)$ with $n > N$ vertices and minimum degree at least ϵn satisfies the following: There exists a partition $\Pi = V_1, \ldots, V_k$ of V such that for every $i \in [k]$ and every $u, v \in V_i$, the number of edge disjoint paths of length at most four from u to v is larger than $8^5 \log n$. Moreover, these sets can be found using a polynomial time algorithm for a fixed ϵ.*

Proof. **(of Lemma 3.6)** First we apply Lemma 3.8 to get the partition Π. Now the proof follows by a simple probabilistic argument. Namely, we color every edge $e \in E$ by choosing one of the colors in $\mathcal{C} = \{a_1, \ldots, a_4, b_1, \ldots, b_4\}$ uniformly and independently at random. Observe that a fixed path P of length at most four is an a-rainbow path with probability at least 8^{-4}. Similarly, P is a b-rainbow path with probability at least 8^{-4}. So any fixed pair $u, v \in V_i$ is not both a-rainbow-connected and b-rainbow-connected with probability at most $2(1-8^{-4})^{8^5 \log n} < n^{-2}$, and therefore the probability that all such pairs are both a-rainbow connected and b-rainbow connected is strictly positive. Hence the desired coloring must exist.

To find the coloring algorithmically, we note that for every *partial* coloring of the edges of the graph it is easy to calculate the *conditional* probability that the fixed pair of vertices u, v is not both a-rainbow-connected and b-rainbow-connected. Therefore we can calculate the conditional expectation of the number of pairs that are not so connected for any partial coloring. Now we can derandomize the random selection of the coloring above by using the conditional expectation method (cf. [2]): In every stage we color one of the remaining edges in a way that does not increase the conditional expectation of the number of unconnected pairs. Since this expectation is smaller than 1 in the beginning, in the end we will have less than 1 unconnected pair, and so all pairs will be connected. ∎

3.5. Proof of Lemma 3.8

Given $\epsilon > 0$ let $L = N_{3.4}(\epsilon, 1)$ and set N to be the smallest number that satisfies $\epsilon^4 \frac{N}{L} > 8^5 \log N$. Now, given any graph $G = (V, E)$ with $n > N$ vertices and minimum degree at least ϵn, we apply Lemma 3.4 with parameters ϵ and 1. Let $\Pi = V_1, V_2, \ldots, V_k$ be the partition of V obtained from Lemma 3.4, while as promised, $k \leq L = N_{3.4}(\epsilon)$.

Fix $i \in [k]$ and $u, v \in V_i$. From Lemma 3.4 we know that there is a component V_a such that u has at least $\frac{\epsilon}{3k} n$ neighbors in V_a. Similarly, there is a component V_b such that v has at least $\frac{\epsilon}{3k} n$ neighbors in V_b. Let $\Gamma_{u,a}$ denote the set of u's neighbors in V_a, and similarly, let $\Gamma_{v,b}$ denote v's neighbors in V_b. We assume in this proof that $V_a \neq V_b$, and at the end it will be clear that the case $V_a = V_b$ can only benefit.

We say that a set $W_u = \{w_1, \ldots, w_t\} \subseteq V_i$ is *distinctly reachable from* u if there are distinct vertices $w'_1, \ldots, w'_t \in \Gamma_{u,a}$ such that for every $j \in [t]$, $\{w_j, w'_j\} \in E$. Notice that the collection of pairs $\{w_j, w'_j\}$ corresponds to a matching in the graph G, where all edges of the matching have one endpoint in V_i and the other endpoint in $\Gamma_{u,a}$. Similarly, we say that $W_v \subseteq V_i$ is distinctly reachable from v if there are distinct vertices $w'_1, \ldots, w'_t \in \Gamma_{v,b}$ such that for every $j \in [t]$, $\{w_j, w'_j\} \in E$.

Observe that it is enough to prove that there exists a set $W \subseteq V_i$ of size $\epsilon^4 \frac{N}{L} > 8^5 \log N$ which is distinctly reachable from both u and v. This will imply the existence of $8^5 \log N$ edge disjoint paths of length four from u to v.

Our first goal is to bound from below the size of the maximal set W_u as above. Since (by Lemma 3.4) V_a and V_i are ϵ^3-regular pairs with density $\geq \frac{\epsilon}{16}$ and since $\epsilon^3 < \epsilon/3$, the number of edges between $\Gamma_{u,a}$ and V_i is at least $\left(\frac{\epsilon}{16} - \epsilon^3\right) |\Gamma_{u,a}| \cdot |V_i|$. Before proceeding, we make the following useful observation.

Observation 3.9. Let $H = (A, B)$ be a bipartite graph with $\gamma|A||B|$ edges. Then H contains a matching M of size $\gamma \frac{|A||B|}{|A|+|B|}$.

Proof. Consider the following process that creates M. Initially $M_0 = \emptyset$. Then in step i, we pick an arbitrary edge $\{a, b\} \in E(H)$, set $M_{i+1} = M_i \cup \{a, b\}$ and remove from $E(H)$ all the edges incident with either a or b. Clearly, in each step the number of removed edges is bounded by $|A| + |B|$, so the process continues for at least $\frac{E(H)}{|A|+|B|} = \gamma \frac{|A||B|}{|A|+|B|}$ steps. Hence $|M| = |\bigcup_i M_i| \geq \gamma \frac{|A||B|}{|A|+|B|}$. ∎

Returning to the proof of Lemma 3.8, by Observation 3.9 the size of a maximal set W_u as above is at least

$$\left(\frac{\epsilon}{16} - \epsilon^3\right) \frac{|\Gamma_{u,a}||V_i|}{|\Gamma_{u,a}| + |V_i|} \geq \left(\frac{\epsilon}{16} - \epsilon^3\right) \frac{\Big(\epsilon n/(3k)\Big)(n/k)}{\epsilon n/(3k) + n/k} \geq \frac{\epsilon^2}{64k} n.$$

To prove that $W = W_u \cap W_a$ is large, we similarly use the regularity condition, but now on the pair $(\Gamma_{v,b}, W_u)$. We get,

$$|E(\Gamma_{v,b}, W_u)| \geq \left(\frac{\epsilon}{16} - \epsilon^3\right) |\Gamma_{v,b}||W_u|.$$

Here too, by Observation 3.9 we can bound from below the size of a maximal matching in the pair $(\Gamma_{v,b}, W_u)$ with

$$\left(\frac{\epsilon}{16} - \epsilon^3\right) \frac{|\Gamma_{v,b}||W_u|}{|\Gamma_{v,b}| + |W_u|} \geq \left(\frac{\epsilon}{16} - \epsilon^3\right) \frac{\left(\frac{\epsilon}{3k} n\right)\left(\frac{\epsilon^2}{64k} n\right)}{\frac{\epsilon}{3k} n + \frac{\epsilon^2}{64k} n} \geq \epsilon^4 \frac{n}{k} \geq \epsilon^4 \frac{n}{L} > 8^5 \log N,$$

where the last inequality follows from our choice of N. Recall that the matching that we found defines the desired set W, concluding the proof. An algorithmic version of this lemma can be derived by simply using an algorithmic version of Lemma 3.4 in the selection of $V_1, \ldots, V_k$ above. ∎

3.6. Graphs with diameter 2

Proof of Theorem 1.4. Consider a random 3-coloring of E, where every edge is colored with one of three possible colors uniformly and independently at random. It is enough to prove that for all pairs $u, v \in V$ the probability that they are not rainbow connected is at most $1/n^2$. Then the proof follows by the union bound (cf. [2]).

Let us fix a pair $u, v \in V$, and bound from above the probability that this pair is not rainbow connected. We know that both $\Gamma(u)$ and $\Gamma(v)$ (the neighborhoods of u and v) contain at least $8 \log n$ vertices.

(1) If $\{u, v\} \in E$ then we are done.
(2) If $|\Gamma(u) \cap \Gamma(v)| \geq 2 \log n$ then there are at least $2 \log n$ edge-disjoint paths of length two from u to v. In this case, the probability that none of these paths is a rainbow path is bounded by $(1/3)^{2 \log n} < 1/n^2$, and we are done.
(3) Otherwise, let $A = \Gamma(u) \setminus \Gamma(v)$ and $B = \Gamma(v) \setminus \Gamma(u)$. We know that $|A|, |B| \geq 6 \log n$, and in addition, since the first two cases do not hold and the diameter of G is two, all the (length two) shortest paths from A's vertices to v go through the vertices in B. This implies that every vertex $x \in A$ has a neighbor $b(x) \in B$ ($b(x)$ need not be a one-one function). Let us consider the set of at least $6 \log n$ edge-disjoint paths $P = \{u, x, b(x) : x \in A\}$. For each $x \in A$, the probability that $u, x, b(x), v$ is a rainbow path (given the color of the edge $(b(x), v)$) is $2/9$. Moreover, this event is independent of the corresponding events for all other members of A, because this proabablity does not change even with full knowledge of the colors of all edges incident with v. Therefore, the probability that none of the paths in P extends to a rainbow path from u to v is at most $(7/9)^{6 \log n} \leq 1/n^2$, as required.

The above proof immediately implies a probabilistic polynomial expected time randomized algorithm with zero error probability (since we can also efficiently check if the coloring indeed makes G 3-rainbow connected). The algorithm can be derandomized and converted to a polynomial time probabilistic algorithm using the method of conditional expectations (cf. [2]) similarly to the proof of Lemma 3.6: For every partial coloring of the edges we can efficiently bound the conditional probability that a fixed pair u, v is not rainbow-connected, using the relevant one of the three cases concerning u and v that were analyzed above. Now we can color the edges one by one, at each time taking care not to increase the bound on the conditional expectation of unconnected pairs that results from the above probability bound for every u and v. Since the bound on the expectation was smaller than 1 before the beginning of the process, in the end we would get a valid 3-rainbow-coloring of G. ∎

4. Concluding remarks and open problems

- Theorem 1.3 asserts that a connected graph with minimum degree at least ϵn has bounded rainbow connectivity. However, the bound obtained is huge as it follows from the Regularity Lemma. It would be interesting to find the "correct" bound. It is even possible that $rc(G) \leq C/\epsilon$ for some absolute constant C.
- The proof of Theorem 1.1 shows that deciding whether $rc(G) = 2$ is NP-Complete. Although this suffices to deduce that computing $rc(G)$ is NP-Hard, we still do not have a proof that deciding whether $rc(G) \leq k$ is NP-Complete for every fixed k. We can easily it for every *even* k by the following reduction from the case $k = 2$. Given a graph G, subdivide every edge into $k/2$ edges. Now, the new graph G' has $rc(G') = k$ if and only if $rc(G) = 2$. Indeed, if $rc(G) = 2$ then take a corresponding red-blue coloring of G and color G' by coloring every subdivided red edge of G with the colors $1, \ldots, k/2$ and every

subdivided blue edge with the colors $k/2+1, \ldots, k$. Conversely, if G' has an edge coloring making it rainbow connected using the colors $1, \ldots, k$, then color each edge e of G as follows. If the subdivision of e contains the color 1, color e red; otherwise, color e blue. This red-blue coloring of G makes G rainbow connected.

It is tempting to conjecture that for every k it is NP-Hard even to distinguish between 2-rainbow-colorable graphs and graphs that are not even k-rainbow-colorable.

- A parameter related to rainbow connectivity is the *rainbow diameter*. In this case we ask for an edge coloring so that for any two vertices, there is a rainbow *shortest* path connecting them. The rainbow diameter number, denoted $rd(G)$ is the smallest number of colors used in such a coloring. Clearly, $rd(G) \geq rc(G)$ and obviously every connected graph with n vertices has $rd(G) < \binom{n}{2}$. Unlike rainbow connectivity, which is a monotone graph property (adding edges never increases the rainbow connectivity number) this is not the case for the rainbow diameter (although we note that constructing an example that proves non-monotonicity is not straightforward). Clearly, computing $rd(G)$ is NP-Hard since $rc(G) = 2$ if and only if $rd(G) = 2$. It would be interesting to prove a version of Theorem 1.3 for rainbow diameter. We conjecture that, indeed, if G is a connected graph with minimum degree at least ϵn then it has a bounded rainbow diameter.
- Suppose that we are given a graph G for which we are *told* that $rc(G) = 2$. Can we rainbow-color it in polynomial time with $o(n)$ colors? For the usual coloring problem, this version has been well studied. It is known that if a graph is 3-colorable (in the usual sense), then there is a polynomial time algorithm that colors it with $\tilde{O}(n^{3/14})$ colors [3].

References

[1] N. Alon, R.A. Duke, H. Lefmann, V. Rödl, and R. Yuster, *The algorithmic aspects of the Regularity Lemma*, Journal of Algorithms 16 (1994), 80–109.

[2] N. Alon and J. H. Spencer, **The Probabilistic Method**, *Second Edition*, Wiley, New York, 2000.

[3] A. Blum and D. Karger, *An $\tilde{O}(n^{3/14})$-coloring algorithm for 3-colorable graphs*, Inform. Process. Lett., 61(1) : 49-53, 1997.

[4] B. Bollobás, **Modern Graph Theory**, Graduate Texts in Mathematics 184, Springer-Verlag 1998.

[5] G. Chartrand, G. L. Johns, K. A. McKeon, and P. Zhang, *Rainbow connection in graphs*, Math. Bohem., 133 (2008), no. 1, 85-98.

[6] Y. Caro, A. Lev, Y. Roditty, Z. Tuza, and R. Yuster, *On rainbow connectivity*, The Electronic Journal of Combinatorics 15(2008), Paper R57.

[7] E. Fischer, A. Matsliah, and A. Shapira, *Approximate Hypergraph Partitioning and Applications*, Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS) 579–589 (2007).

[8] J. Komlós and M. Simonovits, *Szemerédis Regularity Lemma and its applications in graph theory*, In: Combinatorics, Paul Erdös is Eighty (D. Miklós, V. T. Sós, and T. Szönyi, Eds.), Bolyai Society Mathematical Studies, Vol. 2, Budapest (1996), 295-352.

[9] E. Szemerédi Regular partitions of graphs, *Proc. Colloque Inter. CNRS 260*, (CNRS, Paris) 399–401 (1978).

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 255–264
www.stacs-conf.org

NONCLAIRVOYANT SPEED SCALING FOR FLOW AND ENERGY

HO-LEUNG CHAN [1] AND JEFF EDMONDS [2] AND TAK-WAH LAM [3] AND LAP-KEI LEE [3] AND ALBERTO MARCHETTI-SPACCAMELA [4] AND KIRK PRUHS [5]

[1] Max-Planck-Institut für Informatik
E-mail address, H.L. Chan: hlchan@mpi-inf.mpg.de

[2] Department of Computer Science and Engineering, York University
E-mail address, J.Edmonds: jeff@cse.yorku.ca

[3] Department of Computer Science, University of Hong Kong
E-mail address, T.W.Lam: twlam@cs.hku.hk
E-mail address, L.K. Lee: lklee@cs.hku.hk

[4] Dipartimento di Informatica e Sistemistica, Sapienza Università di Roma
E-mail address, A.Marchetti-Spaccamela: alberto@dis.uniroma1.it

[5] Computer Science Department, University of Pittsburgh
E-mail address, K.Pruhs: kirk@cs.pitt.edu

Abstract. We study online nonclairvoyant speed scaling to minimize total flow time plus energy. We first consider the traditional model where the power function is $P(s) = s^\alpha$. We give a nonclairvoyant algorithm that is shown to be $O(\alpha^3)$-competitive. We then show an $\Omega(\alpha^{1/3-\epsilon})$ lower bound on the competitive ratio of any nonclairvoyant algorithm. We also show that there are power functions for which no nonclairvoyant algorithm can be $O(1)$-competitive.

1. Introduction

Energy consumption has become a key issue in the design of microprocessors. Major chip manufacturers, such as Intel, AMD and IBM, now produce chips with dynamically scalable speeds, and produce associated software, such as Intel's SpeedStep and AMD's PowerNow, that enables an operating system to manage power by scaling processor speed. Thus the operating system should have an *speed scaling* policy for setting the speed of the processor, that ideally should work in tandem with a *job selection* policy for determining which job to run. The operating system has dual competing objectives, as it both wants to optimize some schedule quality of service objective, as well as some power related objective.

The work of H.L.Chan was done when he was a postdoc in University of Pittsburgh. T.W.Lam is partially supported by HKU Grant 7176104. A.Marchetti-Spaccamela is partially supported by MIUR FIRB grant RBIN047MH9 and by EU ICT-FET grant 215270 FRONTS. K.Pruhs is partially supported by an IBM faculty award, and from NSF grants CNS-0325353, CCF-0514058, IIS-0534531, and CCF-0830558.

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 255-264
http://drops.dagstuhl.de/opus/volltexte/2009/1815

In this paper, we will consider the objective of minimizing a linear combination of total flow and total energy used. For a formal definitions of the problem that we consider, see subsection 1.2. This objective of flow plus energy has a natural interpretation. Suppose that the user specifies how much improvement in flow, call this amount ρ, is necessary to justify spending one unit of energy. For example, the user might specify that he is willing to spend 1 erg of energy from the battery for a decrease of 5 micro-seconds in flow. Then the optimal schedule, from this user's perspective, is the schedule that optimizes $\rho = 5$ times the energy used plus the total flow. By changing the units of either energy or time, one may assume without loss of generality that $\rho = 1$.

In order to be implementable in a real system, the speed scaling and job selection policies must be online since the system will not in general know about jobs arriving in the future. Further, to be implementable in a generic operating system, these policies must be nonclairvoyant, since in general the operating system does not know the size/work of each process when the process is released to the operating system. All of the previous speed scaling literature on this objective has considered either offline or online clairvoyant policies. In subsection 1.1, we survey the literature on nonclairvoyant scheduling policies for flow objectives on fixed speed processors, and the speed scaling literature for flow plus energy objectives.

Our goal in this paper is to study nonclairvoyant speed scaling assuming an off-line adversary that dynamically chooses the speed of its own machine.

We first analyze the nonclairvoyant algorithm whose job selection policy is Latest Arrival Processsor Sharing (LAPS) and whose speed scaling policy is to run at speed $(1 + \delta)$ times the number of active jobs. LAPS shares the processor equally among the latest arriving constant fraction of the jobs. We adopt the traditional model that the power function, which gives the power as a function of the speed of the processor, is $P(s) = s^\alpha$, where $\alpha > 1$ is some constant. Of particular interest is the case that $\alpha = 3$ since according to the well known cube-root rule, the dynamic power in CMOS based processors is approximately the cube of the speed. Using an amortized local competitiveness argument, we show in section 2 that this algorithm is $O(\alpha^3)$-competitive. The potential function that we use is an amalgamation of the potential function used in [8] for the fixed speed analysis of LAPS, and the potential functions used for analyzing clairvoyant speed scaling policies. This result shows that it is possible for a nonclairvoyant policy to be $O(1)$-competitive if the cube-root rule holds.

It is known that for essentially every power function, there is a 3-competitive *clairvoyant* speed scaling policy [3]. In contrast, we show that the competitiveness achievable by *nonclairvoyant* policies must depend on the power function. In the traditional model, we show in section 3 an $\Omega(\alpha^{1/3-\epsilon})$ lower bound on the competitive ratio of any deterministic nonclairvoyant algorithm. Further, we show in section 3 that there exists a particular power function for which there is no $O(1)$-competitive deterministic nonclairvoyant speed scaling algorithm. The adversarial strategies for these lower bounds are based on the adversarial strategies in [13] for fixed speed processors. Perhaps these lower bound results are not so surprising given the fact that it is known that without speed scaling, resource augmentation is required to achieve $O(1)$-competitiveness for a nonclairvoyant policy [13, 10]. Still a priori it wasn't completely clear that the lower bounds in [13] would carry over. The reason is that in these lower bound instances, the adversary forced the online algorithm into a situation in which the online algorithm had a lot of jobs with a small amount of remaining work, while the adversary had one job left with a lot of remaining work. In the fixed speed setting,

the online algorithm, without resource augmentation, can never get a chance to get rid of this backlog in the face of a steady stream of jobs. However, in a speed scaling setting, one might imagine an online algorithm that speeds up enough to remove the backlog, but not enough to make its energy usage more than a constant time optimal. Our lower bound shows that it is not possible for the online algorithm to accomplish this.

1.1. Related results

We start with some results in the literature about scheduling with the objective of total flow time on a fixed speed processor. It is well known that the online clairvoyant algorithm Shortest Remaining Processing Time (SRPT) is optimal. The competitive ratio of deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized algorithm against an oblivious adversary is $\Omega(\log n)$ [13]. A randomized version of the Multi-Level Feedback Queue algorithm is $O(\log n)$-competitive [11, 5]. The nonclairvoyant algorithm Shortest Elapsed Time First (SETF) is scalable, that is, $(1+\epsilon)$-speed $O(1)$-competitive [10]. SETF shares the processor equally among all jobs that have been run the least. The algorithm Round Robin RR (also called Equipartition and Processor Sharing) that shares the processor equally among all jobs is $(2+\epsilon)$-speed $O(1)$-competitive [7].

Let us first consider the traditional model where the power function is $P = s^\alpha$. Most of the literature assumes the *unbounded speed model*, in which a processor can be run at any real speed in the range $[0, \infty)$. So let us now consider the unbounded speed model. [15] gave an efficient offline algorithm to find the schedule that minimizes average flow subject to a constraint on the amount of energy used, in the case that jobs have unit work. This algorithm can also be used to find optimal schedules when the objective is a linear combination of total flow and energy used. [15] observed that in any locally-optimal schedule, essentially each job i is run at a power proportional to the number of jobs that would be delayed if job i was delayed. [1] proposed the natural online speed scaling algorithm that always runs at a power equal to the number of unfinished jobs (which is lower bound to the number of jobs that would be delayed if the selected job was delayed). [1] did not actually analyze this natural algorithm, but rather analyzed a batched variation, in which jobs that are released while the current batch is running are ignored until the current batch finishes. [1] showed that for unit work jobs this batched algorithm is $O\left(\left(\frac{3+\sqrt{5}}{2}\right)^\alpha\right)$-competitive by reasoning directly about the optimal schedule. [1] also gave an efficient offline dynamic programming algorithm. [4] considered the algorithm that runs at a power equal to the unfinished work (which is in general a bit less than the number of unfinished jobs for unit work jobs). [4] showed that for unit work jobs, this algorithm is 2-competitive with respect to the objective of fractional flow plus energy using an amortized local competitiveness argument. [4] then showed that the natural algorithm proposed in [1] is 4-competitive for total flow plus energy for unit work jobs.

In [4] the more general setting where jobs have arbitrary sizes and arbitrary weights and the objective is weighted flow plus energy has been considered. The authors analysed the algorithm that uses Highest Density First (HDF) for job selection, and always runs at a power equal to the fractional weight of the unfinished jobs. [4] showed that this algorithm is $O(\frac{\alpha}{\log \alpha})$-competitive for fractional weighted flow plus energy using an amortized local competitiveness argument. [4] then showed how to modify this algorithm to obtain an

algorithm that is $O(\frac{\alpha^2}{\log^2 \alpha})$-competitive for (integral) weighted flow plus energy using the known resource augmentation analysis of HDF [6].

Recently, [12] improves on the obtainable competitive ratio for total flow plus energy for arbitrary work and unit weight jobs by considering the job selection algorithm Shortest Remaining Processing Time (SRPT) and the speed scaling algorithm of running at a power proportional to the number of unfinished jobs. [12] proved that this algorithm is $O(\frac{\alpha}{\log \alpha})$-competitive for arbitrary size and unit weight jobs.

In [2] the authors extended the results of [4] for the unbounded speed model to the bounded speed model, where there is an upper bound on the processor speed. The speed scaling algorithm was to run at the minimum of the speed recommended by the speed scaling algorithm in the unbounded speed model and the maximum speed of the processor. The results for the bounded speed model in [2] were improved in [12] proving competitive ratios of the form $O(\frac{\alpha}{\log \alpha})$.

[3] consider a more general model. They assume that the allowable speeds are a countable collection of disjoint subintervals of $[0, \infty)$, and consider arbitrary power functions P that are non-negative, and continuous and differentiable on all but countably many points. They give two main results in this general model. The scheduling algorithm, that uses Shortest Remaining Processing Time (SRPT) for job selection and power equal to one more than the number of unfinished jobs for speed scaling, is $(3 + \epsilon)$-competitive for the objective of total flow plus energy on arbitrary-work unit-weight jobs. The scheduling algorithm, that uses Highest Density First (HDF) for job selection and power equal to the fractional weight of the unfinished jobs for speed scaling, is $(2 + \epsilon)$-competitive for the objective of fractional weighted flow plus energy on arbitrary-work arbitrary-weight jobs.

1.2. Formal Problem Definition and Notations

We study online scheduling on a single processor. Jobs arrive over time and we have no information about a job until it arrives. For each job j, its release time and work requirement (or size) are denoted as $r(j)$ and $p(j)$, respectively. We consider the *nonclairvoyant* model, i.e., when a job j arrives, $p(j)$ is not given and it is known only when j is completed. Preemption is allowed and has no cost; a preempted job can resume at the point of preemption. The processor can vary its speed dynamically to any value in $[0, \infty)$. When running at speed s, the processor processes s units of work per unit time and consumes $P(s) = s^\alpha$ units of energy per unit time, where $\alpha > 1$ is some fixed constant. We call $P(s)$ the *power function*.

Consider any job sequence I and a certain schedule A of I. For any job j in I, the flow time of j, denoted $F_A(j)$, is the amount of time elapsed since it arrives until it is completed. The total flow time of the schedule is $F_A = \sum_{j \in I} F_A(j)$. We can also interpret F_A as follows. Let $n_A(t)$ be the number of jobs released by time t but not yet completed by time t. Then $F_A = \int_0^\infty n_A(t)dt$. Let $s_A(t)$ be the speed of the processor at time t in the schedule. Then the total energy usage of the schedule is $E_A = \int_0^\infty (s(t))^\alpha dt$. The objective is to minimize the sum of total flow time and energy usage, i.e., $F_A + E_A$.

For any job sequence I, a scheduling algorithm ALG needs to specify at any time the speed of the processor and the jobs being processed. We denote ALG(I) as the schedule produced for I by ALG. Let Opt be the optimal offline algorithm such that for any job sequence I, $F_{Opt(I)} + E_{Opt(I)}$ is minimized among all schedules of I. An algorithm ALG is

said to be c-competitive, for any $c \geq 1$, if for all job sequence I,

$$F_{ALG(I)} + E_{ALG(I)} \leq c \cdot (F_{Opt(I)} + E_{Opt(I)})$$

2. An $O(\alpha^3)$-competitive Algorithm

In this section, we give an online nonclairvoyant algorithm that is $O(\alpha^3)$-competitive for total flow time plus energy. We say a job j is *active* at time t if j is released by time t but not yet completed by time t. Our algorithm is defined as follows.

> **Algorithm** LAPS(δ, β)**.** Let $0 < \delta, \beta \leq 1$ be any real. At any time t, the processor speed is $(1+\delta)(n(t))^{1/\alpha}$, where $n(t)$ is the number of active jobs at time t. The processor processes the $\lceil \beta n(t) \rceil$ active jobs with the latest release times (ties are broken by job ids) by splitting the processing speed equally among these jobs.

Our main result is the following.

Theorem 2.1. *When $\delta = \frac{3}{\alpha}$ and $\beta = \frac{1}{2\alpha}$, LAPS$(\delta, \beta)$ is c-competitive for total flow time plus energy, where $c = 4\alpha^3(1 + (1 + \frac{3}{\alpha})^\alpha) = O(\alpha^3)$.*

The rest of this section is devoted to proving Theorem 2.1. We use an amortized local competitiveness argument (see for example [14]). To show that an algorithm is c-competitive it is sufficient to show a potential function such that at any time t the increase in the objective cost of the algorithm plus the change of the potential is at most c times the increase in the objective of the optimum.

For any time t, let $G_a(t)$ and $G_o(t)$ be the total flow time plus energy incurred up to time t by LAPS(δ, β) and the optimal algorithm Opt, respectively. To show that LAPS(δ, β) is c-competitive, it suffices to give a potential function $\Phi(t)$ such that the following four conditions hold.

- *Boundary condition:* $\Phi = 0$ before any job is released and $\Phi \geq 0$ after all jobs are completed.
- *Job arrival:* When a job is released, Φ does not increase.
- *Job completion:* When a job is completed by LAPS(δ, β) or OPT, Φ does not increase.
- *Running condition:* At any other time, the rate of change of G_a plus that of Φ is no more than c times the rate of change of G_o. That is, $\frac{dG_a(t)}{dt} + \frac{d\Phi(t)}{dt} \leq c \cdot \frac{dG_o(t)}{dt}$ during any period of time without job arrival or completion.

Let $n_a(t)$ and $s_a(t)$ be the number of active jobs and the speed in LAPS(δ, β) at time t, respectively. Define $n_o(t)$ and $s_o(t)$ similarly for that of Opt. Then

$$\frac{dG_a(t)}{dt} = \frac{dF_{LAPS}(t)}{dt} + E_{LAPS}(t) = n_a(t) + (s_a(t))^\alpha$$

and, similarly, $\frac{dG_o(t)}{dt} = n_o(t) + (s_o(t))^\alpha$. We define our potential function as follows.

> **Potential function** $\Phi(t)$**.** Consider any time t. For any job j, let $q_a(j,t)$ and $q_o(j,t)$ be the remaining work of j at time t in LAPS(δ, β) and Opt, respectively. Let $\{j_1, \ldots, j_{n_a(t)}\}$ be the set of active jobs in LAPS(δ, β),

ordered by their release time such that $r(j_1) \leq r(j_2) \leq \cdots \leq r(j_{n_a(t)})$. Then,

$$\Phi(t) = \gamma \sum_{i=1}^{n_a(t)} \left(i^{1-1/\alpha} \cdot \max\{0, q_a(j_i,t) - q_o(j_i,t)\} \right)$$

where $\gamma = \alpha(1 + (1 + \frac{3}{\alpha})^\alpha)$. We call $i^{1-1/\alpha}$ the *coefficient* of j_i.

We first check the boundary, job arrival and job completion conditions. Before any job is released or after all jobs are completed, there is no active job in both LAPS(δ, β) and Opt, so $\Phi = 0$ and the boundary condition holds. When a new job j arrives at time t, $q_a(j,t) - q_o(j,t) = 0$ and the coefficients of all other jobs remain the same, so Φ does not change. If LAPS(δ, β) completes a job j, the term for j in Φ is removed. The coefficient of any other job either stays the same or decreases, so Φ does not increase. If Opt completes a job, Φ does not change.

It remains to check the running condition. In the following, we focus on a certain time t within a period of time without job arrival or completion. We omit the parameter t from the notations as t refers only to this certain time. For example, we denote $n_a(t)$ and $q_a(j,t)$ as n_a and $q_a(j)$, respectively. For any job j, if LAPS(δ, β) has processed less than Opt on j at time t, i.e., $q_a(j) - q_o(j) > 0$, then we say that j is a *lagging job* at time t. We start by evaluating $\frac{d\Phi}{dt}$.

Lemma 2.2. *Assume $\delta = \frac{3}{\alpha}$ and $\beta = \frac{1}{2\alpha}$. At time t, if* LAPS(δ, β) *is processing less than $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, then $\frac{d\Phi}{dt} \leq \frac{\gamma}{\alpha} s_o^\alpha + \gamma(1 - \frac{1}{\alpha}) n_a$. Else if* LAPS$(\delta, \beta)$ *is processing at least $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, then $\frac{d\Phi}{dt} \leq \frac{\gamma}{\alpha} s_o^\alpha - \frac{\gamma}{\alpha} n_a$.*

Proof. We consider $\frac{d\Phi}{dt}$ as the combined effect due to the processing of LAPS(δ, β) and Opt. Note that for any job j, $q_a(j)$ is decreasing at a rate of either 0 or $-s_a/\lceil \beta n_a \rceil$. Thus the rate of change of Φ due to LAPS(δ, β) is non-positive. Similarly, the rate of change of Φ due to Opt is non-negative.

We first bound the rate of change of Φ due to Opt. The worst case is that Opt is processing the job with the largest coefficient, i.e., $n_a^{1-1/\alpha}$. Thus the rate of change of Φ due to Opt is at most $\gamma n_a^{1-1/\alpha}(-\frac{dq_o(j_{n_a})}{dt}) = \gamma n_a^{1-1/\alpha} s_o$. We apply Young's Inequality [9], which is formally stated in Lemma 2.3, by setting $f(x) = x^{\alpha-1}$, $f^{-1}(x) = x^{1/(\alpha-1)}$, $g = s_o$ and $h = n_a^{1-1/\alpha}$. Then, we have

$$s_o n_a^{1-1/\alpha} \leq \int_0^{s_o} x^{\alpha-1} dx + \int_0^{n_a^{1-1/\alpha}} x^{1/(\alpha-1)} dx = \frac{1}{\alpha} s_o^\alpha + (1 - \frac{1}{\alpha}) n_a$$

If LAPS(δ, β) is processing less than $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, we just ignore the effect due to LAPS(δ, β) and take the bound that $\frac{d\Phi}{dt} \leq \frac{\gamma}{\alpha} s_o^\alpha + \gamma(1 - \frac{1}{\alpha}) n_a$.

If LAPS(δ, β) is processing at least $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ lagging jobs, let j_i be one of these lagging jobs. We notice that j_i is among the $\lceil \beta n_a \rceil$ active jobs with the latest release times. Thus, the coefficient of j_i is at least $(n_a - \lceil \beta n_a \rceil + 1)^{1-1/\alpha}$. Also, j_i is being processed at a speed of $s_a/\lceil \beta n_a \rceil$, so $q_a(j_i, t)$ is decreasing at this rate. LAPS(δ, β) is processing at least $(1 - \frac{1}{2\alpha})\lceil \beta n_a \rceil$ such lagging jobs, so the rate of change of Φ due to LAPS(δ, β) is more

negative than

$$\begin{aligned} & \gamma\left((1-\frac{1}{2\alpha})\lceil\beta n_a\rceil\right)(n_a-\lceil\beta n_a\rceil+1)^{1-1/\alpha}\left(\frac{-s_a}{\lceil\beta n_a\rceil}\right) \\ \leq\ & -\gamma(1-\frac{1}{2\alpha})(n_a-\beta n_a)^{1-1/\alpha}(s_a) && (\text{since } -\lceil\beta n_a\rceil+1\geq-\beta n_a) \\ \leq\ & -\gamma(1-\frac{1}{2\alpha})(1-\beta)(1+\delta)n_a && (\text{since } s_a=(1+\delta)n_a^{1/\alpha}) \end{aligned}$$

When $\beta=\frac{1}{2\alpha}$ and $\delta=\frac{3}{\alpha}$, simple calculation shows that $(1-\frac{1}{2\alpha})(1-\beta)(1+\delta)\geq 1$ and hence the last term above is at most $-\gamma n_a$. It follows that $\frac{d\Phi}{dt}\leq\frac{\gamma}{\alpha}s_o^\alpha+\gamma(1-\frac{1}{\alpha})n_a-\gamma n_a=\frac{\gamma}{\alpha}s_o^\alpha-\frac{\gamma}{\alpha}n_a$. ∎

Below is the formal statement of Young's Inequality, which is used in the proof of Lemma 2.2.

Lemma 2.3 (Young's Inequality [9])**.** *Let f be any real-value, continuous and strictly increasing function f such that $f(0)=0$. Then, for all $g,h\geq 0$, $\int_0^g f(x)dx+\int_0^h f^{-1}(x)dx\geq gh$, where f^{-1} is the inverse function of f.*

We are now ready to show the following lemma about the running condition.

Lemma 2.4. *Assume $\delta=\frac{3}{\alpha}$ and $\beta=\frac{1}{2\alpha}$. At time t, $\frac{dG_a}{dt}+\frac{d\Phi}{dt}\leq c\cdot\frac{dG_o}{dt}$, where $c=4\alpha^3(1+(1+\frac{3}{\alpha})^\alpha)$.*

Proof. We consider two cases depending on the number of lagging jobs that LAPS(δ,β) is processing at time t. If LAPS(δ,β) is processing at least $(1-\frac{1}{\alpha})\lceil\beta n_a\rceil$ lagging jobs, then

$$\begin{aligned} \frac{dG_a}{dt}+\frac{d\Phi}{dt} &= n_a+s_a^\alpha+\frac{d\Phi}{dt} \\ &\leq n_a+(1+\delta)^\alpha n_a+\frac{\gamma}{\alpha}s_o^\alpha-\frac{\gamma}{\alpha}n_a && (\text{by Lemma 2.2}) \\ &= (1+(1+\delta)^\alpha-\frac{\gamma}{\alpha})n_a+\frac{\gamma}{\alpha}s_o^\alpha \end{aligned}$$

Since $\delta=\frac{3}{\alpha}$ and $\gamma=\alpha(1+(1+\frac{3}{\alpha})^\alpha)$, the coefficient of n_a becomes zero and $\frac{dG_a}{dt}+\frac{d\Phi}{dt}\leq\frac{\gamma}{\alpha}s_o$. Note that $\frac{\gamma}{\alpha}=(1+(1+\frac{3}{\alpha})^\alpha)\leq c$ and $\frac{dG_o}{dt}=n_o+s_o^\alpha$, so we have $\frac{dG_a}{dt}+\frac{d\Phi}{dt}\leq c\cdot\frac{dG_o}{dt}$.

If LAPS(δ,β) is processing less than $(1-\frac{1}{2\alpha})\lceil\beta n_a\rceil$ lagging jobs, the number of jobs remaining in Opt is $n_o\geq\lceil\beta n_a\rceil-(1-\frac{1}{2\alpha})\lceil\beta n_a\rceil=\frac{1}{2\alpha}\lceil\beta n_a\rceil\geq\frac{1}{2\alpha}\beta n_a=\frac{1}{4\alpha^2}n_a$. Therefore,

$$\begin{aligned} \frac{dG_a}{dt}+\frac{d\Phi}{dt} &= n_a+s_a^\alpha+\frac{d\Phi}{dt} \\ &\leq n_a+(1+\delta)^\alpha n_a+\frac{\gamma}{\alpha}s_o^\alpha+\gamma(1-\frac{1}{\alpha})n_a && (\text{by Lemma 2.2}) \\ &= (1+(1+\delta)^\alpha+\gamma(1-\frac{1}{\alpha}))n_a+\frac{\gamma}{\alpha}s_o^\alpha \\ &\leq 4\alpha^2(1+(1+\delta)^\alpha+\gamma(1-\frac{1}{\alpha}))n_o+\frac{\gamma}{\alpha}s_o^\alpha \end{aligned}$$

Since $\delta=\frac{3}{\alpha}$ and $\gamma=\alpha(1+(1+\frac{3}{\alpha})^\alpha)$, the coefficient of n_o becomes $4\alpha^3(1+(1+\frac{3}{\alpha})^\alpha)=c$. The coefficient of s_o^α is $(1+(1+\frac{3}{\alpha})^\alpha)\leq c$. Since $\frac{dG_o(t)}{dt}=n_o+s_o^\alpha$, we obtain $\frac{dG_a(t)}{dt}+\frac{d\Phi}{dt}\leq c\cdot\frac{dG_o(t)}{dt}$. Note that this case is the bottleneck leading to the current competitive ratio. ∎

Combining Lemma 2.4 with the discussion on the boundary, job arrival and job completion conditions, Theorem 2.1 follows.

3. Lower Bounds

In this section, we show that every nonclairvoyant algorithm is $\Omega(\alpha^{1/3-\epsilon})$-competitive in the traditional model where the power function $P(s) = s^\alpha$. We further extend the lower bound to other power functions P and show that for some power function, any algorithm is $\omega(1)$-competitive. We first prove the following lemma.

Lemma 3.1. *Let $P(s)$ be any non-negative, continuous and super-linear power function. Let $k, v \geq 1$ be any real such that $P(v) \geq 1$. Then, any algorithm is $\Omega(\min\{k, P(v + \frac{1}{16(kP(v))^3})/P(v)\})$-competitive.*

Proof. Let ALG be any algorithm and Opt be the offline adversary. Let $n = \lceil kP(v) \rceil$. We release n jobs $j_1, j_2, \ldots, j_n$ at time 0. Let T be the first time that some job in ALG is processed for at least n units of work. Let $G(T)$ be the total flow time plus energy incurred by ALG up to T. We consider two cases depending on $G(T) \geq kn^3$ or $G(T) < kn^3$. If $G(T) \geq kn^3$, Opt reveals that all jobs are of size n. By running at speed 1, Opt completes all jobs by time n^2. The total flow time plus energy of Opt is at most $n^3 + n^2 P(1) \leq 2n^3$, so ALG is $\Omega(k)$-competitive.

The rest of the proof assumes $G(T) < kn^3$. Let $q_1, q_2, \ldots, q_n$ be the amount of work ALG has processed for each of the n jobs. Without loss of generality, we assume $q_n = n$. Opt reveals that the size of each job j_i is $p_i = q_i + 1$. Thus, at time T, ALG has n remaining jobs, each of size 1. For Opt, it runs at the same speed as ALG during $[0, T]$ and processes exactly the same job as ALG except on j_n. By distributing the n units of work processed on j_n to all the n jobs, Opt can complete $j_1, \ldots, j_{n-1}$ by time T and the remaining size of j_n is n. As Opt is simulating ALG on all jobs except j_n, the total flow plus energy incurred by Opt up to T is at most $G(T) < kn^3$.

During $[T, T + n^4]$, Opt releases a stream of small jobs. Specifically, let $\epsilon < \frac{1}{n^5 v^2}$ be any real. For $i = 1, \ldots, \frac{n^4}{\epsilon}$, a small job j_i' is released at $T + (i-1)\epsilon$ with size ϵv. Opt can run at speed v and complete each small job before the next one is released. Thus, Opt has at most one small job and j_n remaining at any time during $[T, T + n^4]$. The flow time plus energy incurred during this period is $2n^4 + n^4 P(v)$. Opt can complete j_n by running at speed 1 during $[T+n^4, T+n^4+n]$, incurring a cost of $n+nP(1)$. Thus, the total flow time plus energy of Opt for the whole job sequence is at most $kn^3 + 2n^4 + n^4 P(v) + n + nP(1) = O(n^4 P(v))$.

For ALG, we first show that its total work done on the small jobs during $[T, T + n^4]$ is at least $n^4 v - 1$. Otherwise, there are at least $\frac{1}{\epsilon v} > n^5 v$ small jobs not completed by $T + n^4$. The best case is when these jobs are released during $[T + n^4 - \frac{1}{v}, T + n^4]$ and their total flow time incurred is $\Omega(n^5)$. It means that ALG is $\Omega(k)$-competitive as $n = \lceil kP(v) \rceil$.

We call $j_1, \ldots, j_n$ *big* jobs and then consider the number of big jobs completed by ALG by time $T + n^4$. If ALG completes less that $\frac{1}{2}n + 1$ big jobs by time $T + n^4$, then ALG has at least $\frac{1}{2}n - 1$ big jobs remaining at any time during $[T, T + n^4]$. The total flow time of ALG is at least $\Omega(n^5)$, meaning that ALG is $\Omega(k)$-competitive. If ALG completes at least $\frac{1}{2}n + 1$ big jobs by time $T + n^4$, the total work done by ALG during $[T, T + n^4]$ is at least

$n^4v - 1 + \frac{1}{2}n + 1$. The total energy used by ALG is at least

$$P(\frac{n^4v + \frac{1}{2}n}{n^4}) \times n^4 = P(v + \frac{1}{2n^3}) \times n^4 \geq P(v + \frac{1}{16(kP(v))^3}) \times n^4$$

The last inequality comes from the fact that $n = \lceil kP(v) \rceil \leq 2kP(v)$. Hence, ALG is at least $\Omega(P(v + \frac{1}{16(kP(v))^3})/P(v))$-competitive. ∎

Then, we can apply Lemma 3.1 to obtain the lower bound for the power function $P(s) = s^\alpha$.

Theorem 3.2. *When the power function is $P(s) = s^\alpha$ for some $\alpha > 1$, any algorithm is $\Omega(\alpha^{1/3-\epsilon})$-competitive for any $0 < \epsilon < 1/3$.*

Proof. We apply Lemma 3.1 by putting $k = \alpha^{1/3-\epsilon}$ and $v = 1$. Then, $P(v) = 1$ and

$$P(v + \frac{1}{16(kP(v))^3})/P(v) = \left(1 + \frac{1}{16(\alpha^{1/3-\epsilon})^3}\right)^\alpha = \left(1 + \frac{1}{16\alpha^{1-3\epsilon}}\right)^{(\alpha^{1-3\epsilon}) \times \alpha^{3\epsilon}}$$

Since $(1+\frac{1}{16x})^x$ is increasing with x and $\alpha^{1-3\epsilon} \geq 1$, the last term above is at least $(1+\frac{1}{16})^{\alpha^{3\epsilon}}$. Thus, $\min\{k, P(v+\frac{1}{16(kP(v))^3})/P(v)\} \geq \min\{\alpha^{1/3-\epsilon}, (\frac{17}{16})^{\alpha^{3\epsilon}}\} = \Omega(\alpha^{1/3-\epsilon})$, and the theorem follows. ∎

We also show that for some power function, any algorithm is $\omega(1)$-competitive.

Theorem 3.3. *There exists some power function P such that any algorithm is $\omega(1)$-competitive.*

Proof. We want to find a power function P such that for any $k \geq 1$, there exists a speed v such that $P(v + \frac{1}{16(kP(v))^3})/P(v) \geq k$. Then by setting k and v correspondingly to Lemma 3.1, any algorithm is at least k-competitive for any $k \geq 1$. It implies that any algorithm is $\omega(1)$-competitive. For example, consider the power function

$$P(s) = \frac{1}{(4(2-s))^{1/4}} \ , \ 0 \leq s < 2$$

Let P' be the derivative of P. We can verify that $P'(s) = (P(s))^5$ for all $0 \leq s < 2$. For any k, let $v \geq 1$ be a speed such that $P(v) \geq 16k^4$. Then,

$$P(v + \frac{1}{16(kP(v))^3}) \geq P(v) + P'(v)\frac{1}{16(kP(v))^3} \geq (P(v))^5\frac{1}{16(kP(v))^3} \geq kP(v)$$

Thus, $P(v + \frac{1}{16(kP(v))^3})/P(v) \geq k$ and the theorem follows. ∎

4. Conclusion

We show that nonclairvoyant policies can be $O(1)$-competitive in the traditional power model. However, we showed that in contrast to the case for clairvoyant algorithms, there are power functions that are sufficiently quickly growing that nonclairvoyant algorithms can not be $O(1)$-competitive.

One obvious open problem is to reduce the competitive ratio achievable by a nonclairvoyant algorithm in the case that the cube-root rule holds to something significantly more reasonable than the rather high bound achieved here.

The standard/best nonclairvoyant job selection policy for a fixed speed processor is Short Elapsed Time First (SETF). The most obvious candidate speed scaling policy would be to use SETF for job selection, and to run at power somewhat higher than the number of active jobs. The difficulty with analyzing this speed scaling algorithm is that it is hard to find potential functions that interact well with SETF. It would be interesting to provide an analysis of this algorithm.

References

[1] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

[2] N. Bansal, H.L. Chan, T.W. Lam, and L.K. Lee. Scheduling for bounded speed processors. In *Proc. of International Colloquium on Automata, Languages and Programming, ICALP*, pages 409 – 420, 2008.

[3] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. submitted.

[4] Nikhil Bansal, Kirk Pruhs, and Cliff Stein. Speed scaling for weighted flow time. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 805–813, 2007.

[5] Luca Becchetti and Stefano Leonardi. Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines. *J. ACM*, 51(4):517–539, 2004.

[6] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. Online weighted flow time and deadline scheduling. *J. Discrete Algorithms*, 4(3):339–352, 2006.

[7] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.

[8] Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. Manuscript.

[9] G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1952.

[10] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[11] Bala Kalyanasundaram and Kirk Pruhs. Minimizing flow time nonclairvoyantly. *J. ACM*, 50(4):551–567, 2003.

[12] T.W. Lam, L.K. Lee, Isaac To, and P. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proc. of European Symposium on Algorithms, ESA*, 2008, to appear.

[13] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theorertical Computer Scienc3*, 130(1):17–47, 1994.

[14] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[15] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3), 2008.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 265–276
www.stacs-conf.org

AN APPROXIMATION ALGORITHM FOR l_∞-FITTING ROBINSON STRUCTURES TO DISTANCES

V. CHEPOI [1] AND M. SESTON [1]

[1] LIF, Université d'Aix-Marseille, Marseille Cedex 9, France
E-mail address: chepoi,seston@lif.univ-mrs.fr

Abstract. In this paper, we present a factor 16 approximation algorithm for the following NP-hard distance fitting problem: given a finite set X and a distance d on X, find a Robinsonian distance d_R on X minimizing the l_∞-error $||d-d_R||_\infty = \max_{x,y\in X}\{|d(x,y)-d_R(x,y)|\}$. A distance d_R on a finite set X is Robinsonian if its matrix can be symmetrically permuted so that its elements do not decrease when moving away from the main diagonal along any row or column. Robinsonian distances generalize ultrametrics, line distances and occur in the seriation problems and in classification.

1. Introduction

1.1. Seriation problem. Many applied algorithmic problems involve ordering of a set of objects so that closely coupled objects are placed near each other. These problems occur in such diverse applications as data analysis, archeological dating, numerical ecology, matrix visualization methods, DNA sequencing, overlapping clustering, graph linear arrangement, and sparse matrix envelope reduction. For example, a major issue in classification and data analysis is to visualize simple geometrical and relational structures between objects. Necessary for such an analysis is a dissimilarity on a set of objects, which is measured directly or computed from a data matrix. The classical *seriation problem* [16, 18] consists in finding of a simultaneous permutation of the rows and the columns of the dissimilarity matrix with the objective of revealing an underlying one-dimensional structure. The basic idea is that small values should be concentrated around the main diagonal as closely as possible, whereas large values should fall as far from it as possible. This goal is best achieved by considering the so-called *Robinson property* [20]: a dissimilarity matrix has this property if its values do not decrease when moving away from the main diagonal along any row or column. Experimental data usually contain errors, whence the dissimilarity can be measured only approximatively. As a consequence, any simultaneous permutation of the rows and the columns of the dissimilarity matrix gives a matrix which fails to satisfy the Robinson property, and we are led to the problem of finding a matrix reordering which is as close as

1998 ACM Subject Classification: Primary 68W25; Secondary 62H30 and 62-07.
Key words and phrases: Robinsonian dissimilarity, approximation algorithm, fitting problem.
We are grateful to Bernard Fichet for numerous insightful discussions during the work on this paper.
The authors were partly supported by the ANR grant BLAN06-1-138894 (projet OPTICOMB).

possible to a Robinson matrix. As an error measure one can use the l_p-distance between two matrices. Several heuristics for seriation using Robinson matrices have been considered in the literature (the package seriation [14] contains their implementation). However, these methods either have exponential complexity or do not provide any optimality guarantee of the obtained solutions. In this paper, we provide a factor 16 algorithm for the NP-hard problem of optimally fitting a dissimilarity matrix by a Robinson matrix under the l_∞-error.

1.2. Definitions and the problem. Let X be a set of n elements to sequence, endowed with a *dissimilarity function* $d : X^2 \to \mathbb{R}^+ \cup \{0\}$ (i.e., $d(x, y) = d(y, x) \geq 0$ and $d(x, x) = 0$). A dissimilarity d and a total order $\prec$ on X are *compatible* if $d(x, y) \geq d(u, v)$ for any four elements such that $x \prec u \prec v \prec y$. Then d is *Robinsonian* if it admits a compatible order. Basic examples of Robinson dissimilarities are the *ultrametrics* and the standard *line-distance* between n points on the line. Denote by $\mathcal{D}$ and $\mathcal{R}$ the sets of all dissimilarities and of all Robinson dissimilarities on X. For $d, d' \in \mathcal{D}$, define the l_∞-error by $||d - d'||_\infty = \max_{x,y\in X}\{|d(x, y) - d'(x, y)|\}$. To formulate the corresponding fitting problem, we relax the notions of compatible order and Robinson dissimilarity. Given $\epsilon \geq 0$, a total order $\prec$ on X is called ϵ-*compatible* if $x \prec u \prec v \prec y$ implies $d(x, y) + 2\epsilon \geq d(u, v)$. An ϵ-*Robinsonian dissimilarity* is a dissimilarity admitting an ϵ-compatible order, i.e., for each pair $x, y \in X$ one can pick a value $d_R(x, y) \in [d(x, y) - \epsilon, d(x, y) + \epsilon]$ so that the resulting dissimilarity d_R is Robinsonian. In this paper, we study the following NP-hard [8] optimization problem:

Problem l_∞-FITTING-BY-ROBINSON: *Given $d \in \mathcal{D}$, find a Robinson dissimilarity $d_R \in \mathcal{R}$ minimizing the l_∞-error $||d - d_R||_\infty$, i.e., find a least ϵ such that d is ϵ-Robinsonian.*

1.3. Related work. Fitting general distances by simpler distances (alias low-distortion embeddings) is a classical problem in mathematics, data analysis, phylogeny, and, more recently, in computer science. We review here only the results about l_∞-fitting of distances (this error measure is also known as the *maximum additive distortion* or the *maximum additive two-sided error* [5]). Farach et al. [13] showed that l_∞-fitting of a distance d by an ultrametric is polynomial. This result has been used by Agarwala et al. [1] to design a factor 3 approximation algorithm for l_∞-fitting of distances by tree-distances, a problem which has been shown to be strongly NP-hard [1]. A unified and simplified treatment of these results of [1, 13] using sub-dominants was given in [7]. A factor 2 approximation algorithm for the NP-hard problem of l_∞-fitting of a dissimilarity by a line-distance was given by Hstad et al. [15]. Bădoiu [4] proposed a constant-factor algorithm for l_∞-fitting of distances by l_1-distances in the plane.

Seriation is important in archeological dating, clustering hypertext orderings, numerical ecology, sparse matrix ordering, matrix visualization methods, and DNA sequencing [3, 6, 16, 18, 19, 20]. A package seriation implementing various seriation methods is described in [14]. The most common methods for clustering provide a visual display of data in the form of dendrograms. Dissimilarities in perfect agreement with dendrograms (i.e., ultrametrics) are Robinsonian. Generalizing this correspondence, [11, 12] establish that the Robinson dissimilarities can be visualized by hierarchical structures called pyramids.

1.4. Our result and techniques. The main result of the paper is a factor 16 approximation algorithm for the problem l_∞-FITTING-BY-ROBINSON. The basic setting of our algorithm goes as follows. First we show that the optimal error ϵ^* belongs to a well-defined list Δ of size $O(n^4)$. As in some other minmax problems, our approximation algorithm tests the entries of Δ, using a parameter ϵ, which is the "guess" for ϵ^*. For current $\epsilon \in \Delta$, the

algorithm either finds that no ϵ-compatible order exist, in which case the input dissimilarity d is not ϵ-Robinsonian, or it returns a 16ϵ-compatible order. Now, if ϵ is the least value for which the algorithm does not return the negative answer, then $\epsilon^* \geq \epsilon$, and the returned 16ϵ-Robinsonian dissimilarity has l_∞-error at most $16\epsilon^*$, establishing that we have a factor 16 approximation algorithm.

For $\epsilon \in \Delta$, a canonical binary relation $\preccurlyeq$ is computed so that any ϵ-compatible total order refines $\preccurlyeq$ or its dual. If $\preccurlyeq$ is not a partial order, then the algorithm halts and returns the negative answer. If $\preccurlyeq$ is a total order, then we are done. Otherwise, we select a maximal chain $P = (a_1, a_2, \ldots, a_p)$ of the partial order $\preccurlyeq$ and search to fit each element of $X^\circ := X \setminus P$ between two consecutive elements of P. We say that $a_i, a_{i+1} \in P$ form a *hole* H_i and that all elements $x \in X^\circ$ assigned between a_i and a_{i+1} are *located* in H_i. This distribution of the elements to holes is performed so that (a) all elements X_i of X° located in the same hole H_i must "fit" in this hole, i.e., for all $x, y \in X_i$ one of the orders $a_i \prec x \prec y \prec a_{i+1}$ or $a_i \prec y \prec x \prec a_{i+1}$ must be $c\epsilon$-compatible for some $c \leq 12$. Partitioning X° into sets X_i, $i = 1, \ldots, p-1$, is not obvious. Even if such a partition is available, we cannot directly apply a recursive call to each X_i, because (b) the elements located outside the hole H_i will impose a certain order on the elements of X_i and, since we tolerate some errors, (c) we cannot ensure that X_i is exactly the set of elements which must be located in H_i in some ϵ-compatible total order. To deal with (a), we give a classification of admissible and pairwise admissible holes for elements of X°. This allows to show that, if we tolerate a 12ϵ-error, then each element $x \in X^\circ$ can be located in the leftmost or rightmost admissible hole for x (we call them *bounding holes* of x). Both locations are feasible unless several elements have the same pair of bounding holes. For $i < j$, let X_{ij} be the set of all elements of X° having H_i and H_{j-1} as bounding holes. To deal with (b) and (c), on each set X_{ij} we define a directed graph $\mathcal{L}^{\rightarrow}_{ij}$. The strongly connected components (which we call *cells*) of $\mathcal{L}^{\rightarrow}_{ij}$ have the property that in any ϵ-compatible order all elements of the same component must be located in the same hole. In fact the cells (and not the sets X_i) are the units to which we apply the recursive calls in the algorithm. To decide in which hole H_i or H_{j-1} to locate each cell of $\mathcal{L}^{\rightarrow}_{ij}$ and to define the relative order between the cells assigned to the same hole, we define another directed graph $\mathcal{G}_{ij}$ whose vertices are the cells of $\mathcal{L}^{\rightarrow}_{ij}$ in such a way that (i) if some $\mathcal{G}_{ij}$ does not admit a partition into two acyclic subgraphs then no ϵ-compatible order exist and (ii) if $\mathcal{G}_{ij}$ has a partition into two acyclic subgraphs $\mathcal{G}^-_{ij}$ and $\mathcal{G}^+_{ij}$, then all cells of $\mathcal{G}^-_{ij}$ will be located in H_i, all cells of $\mathcal{G}^+_{ij}$ will be located in H_{j-1}, and the topological ordering of each of these graphs defines the relative order between the cells. To partition $\mathcal{G}_{ij}$ into two acyclic subgraphs (this problem in general is NP-complete [17]), we investigate the specific properties of graphs in question, allowing us to define a 2-SAT formula Φ_{ij} which is satisfiable if and only if the required bipartition of $\mathcal{G}_{ij}$ exists. Finally, to locate in each hole H_i the cells coming from different subgraphs $\mathcal{G}^+_{j'i}, \mathcal{G}^-_{ij}$, and $\mathcal{G}^-_{ij''}$ with $j' < i < j < j''$, we use the following separation rule: the cells of $\mathcal{G}^+_{j'i}$ are located to the left of the cells of $\mathcal{G}^-_{ij}$ and the cells of $\mathcal{G}^-_{ij}$ are located to the right of the cells of $\mathcal{G}^-_{ij''}$. Due to space constraints, all missing proofs are given in the full version [9].

2. Preliminary results

The $\prec$-restricted problem is obtained from l_∞-FITTING-BY-ROBINSON by fixing the total order $\prec$ on X. Let $\check{d}_\prec$ be a dissimilarity defined by setting $\check{d}_\prec(x, y) = \max\{d(u, v) : x \prec u \prec v \prec y\}$ for all $x, y \in X$ with $x \prec y$ (we suppose here that $a \prec a$ for any $a \in X$).

Let $2\tilde{\epsilon}_\prec = ||d - \check{d}_\prec||_\infty$ and let $\tilde{d}_\prec$ be the (Robinsonian) dissimilarity obtained from $\check{d}_\prec$ by setting $\tilde{d}_\prec(x,y) = \max\{\check{d}_\prec(x,y) - \tilde{\epsilon}_\prec, 0\}$ for all $x, y \in X, x \neq y$. Then, the following holds:

Proposition 2.1. *For a total order $\prec$ on X and $d \in \mathcal{D}$, $\tilde{d}_\prec$ minimizes $||d - d'||_\infty$.*

Proposition 2.1 establishes that an optimal solution of the problem l_∞-FITTING-BY-ROBINSON can be selected among $n!$ Robinsonian dissimilarities of the form $\tilde{d}_\prec$. In the full version, we show that the natural heuristic similar to the factor 3 approximation algorithms of Håstad et al. [15] and Agarwala et al. [1] (which instead of $n!$ total orders considers only n orders) does not provide a constant-factor approximation algorithm for our problem. Proposition 2.1 also implies that the optimal error ϵ^* in l_∞-FITTING-BY-ROBINSON belongs to a well-defined list $\Delta = \{\frac{1}{2}|d(x,y) - d(x',y')| : x, y, x', y' \in X\}$ of size $O(n^4)$.

Given $d \in \mathcal{D}$ and $\epsilon \in \Delta$, we define a partial order $\preccurlyeq$ such that every ϵ-compatible total order $\prec$ refines either $\preccurlyeq$ or its dual. For this, we set $p \preccurlyeq q$ for two arbitrary elements $p, q \in X$, and close $\preccurlyeq$ using the properties of partial orders and the following observation: *if $d(x,y) > \max\{d(x,z), d(z,y)\} + 2\epsilon$, then in all ϵ-compatible with d orders z must be located between x and y.* In this case, if we know that two of the elements x, z, y are in relation $\preccurlyeq$ then we can extend this relation to the whole triplet. For example, if we know that $x \preccurlyeq z$, then we conclude that also $z \preccurlyeq y$ and $x \preccurlyeq z$. If the resulting $\preccurlyeq$ is not a partial order, then d does not admit an ϵ-compatible total order. So, further let $\preccurlyeq$ be a partial order. For two disjoint subsets A, B of X, set $A \preccurlyeq B$ if $a \preccurlyeq b$ for any $a \in A$ and $b \in B$. We write $x?y$ if neither $x \preccurlyeq y$ nor $y \preccurlyeq x$ hold. For two numbers α and β we will use the following notations (i) $\alpha \approx_c \beta$ if $|\alpha - \beta| \leq c\epsilon$, (ii) $\beta \gtrsim_c \alpha$ if $\beta \geq \alpha - c\epsilon$, and (iii) $\beta \gg_c \alpha$ if $\beta > \alpha + c\epsilon$. We continue with basic properties of the canonical partial order $\preccurlyeq$: *If $w \preccurlyeq \{v, z\}$, $v?z$, $u \preccurlyeq v$, $u?z$, and $w?u$, then:* (i) $d(v,w) \approx_2 d(z,w)$; (ii) $d(v,z) \lesssim_2 \min\{d(v,w), d(z,w)\}$; (iii) $d(w,z) \approx_4 \{d(u,v), d(u,z)\}$; (iv) $d(w,u) \lesssim_2 \min\{d(w,v), d(u,v)\}$.

3. Pairwise admissible holes

3.1. Admissible holes. Let $P = (a_1, a_2, \ldots, a_{p-1}, a_p)$ be a maximal chain of the partial order $\preccurlyeq$. For notational convenience, we assume that all elements of X° must be located between a_1 and a_p (a_1 and a_p can be artificially added); this way, every element of X° must be located in a hole. Let H_{ij} be the union of all holes comprised between a_i, a_j. For $x \in X^\circ$, denote by $H(x)$ the union of all holes H_i such that $x?a_i$ or $x?a_{i+1}$. If $H(x) = H_{ij}$, the holes H_i and H_{j-1} are called *bounding holes*; see Fig. 1 (note that $a_i = \max\{a_k \in P : a_k \preccurlyeq x\}$ and $a_j = \min\{a_k \in P : x \preccurlyeq a_k\}$ for $x \in X^\circ$). All other holes of $H(x)$ are called *inner holes*. Since $x \notin P$, $H(x)$ contains at least two holes. The hole H_k of $H(x)$ is *x-admissible,* if the total order on $P \cup \{x\}$ obtained from $\preccurlyeq$ by adding the relation $a_k \preccurlyeq x \preccurlyeq a_{k+1}$ is ϵ-compatible with d. It can be easily shown that the bounding holes of $H(x)$ must be x-admissible. Denote by d_x the mean value of $\min\{d(x,a_k) : i < k < j\}$ and $\max\{d(x,a_k) : i < k < j\}$. We call $\delta_k = d(a_k, a_{k+1})$ the *size* of the hole H_k. Then the following holds:

Lemma 3.1. *If an inner hole H_k of $H(x)$ is x-admissible, then $d_x \approx_1 \{d(x,a_k), d(x,a_{k+1})\} \approx_2 \delta_k$. In particular, $\delta_k \approx_3 d_x$. More generally, for all $k, k' \in]i, j[$, we have $d_x \gtrsim_3 d(a_k, a_{k'})$.*

3.2. Pairwise admissible holes. A pair $\{H_k, H_{k'}\}$ of holes is called *(x, y, c)-admissible* if H_k is x-admissible, $H_{k'}$ is y-admissible, and the total order on $P \cup \{x, y\}$ obtained by adding to $\preccurlyeq$ the relations $a_k \preccurlyeq x \preccurlyeq a_{k+1}$ and $a_{k'} \preccurlyeq y \preccurlyeq a_{k'+1}$ is $c\epsilon$-compatible. Denote by $AH(x)$ the set of all x-admissible holes H_k so that for each $y \in X^\circ$, $y \neq x$, there exists an y-admissible hole $H_{k'}$ such that $\{H_k, H_{k'}\}$ is a $(x, y, 1)$-admissible pair. Further we can assume

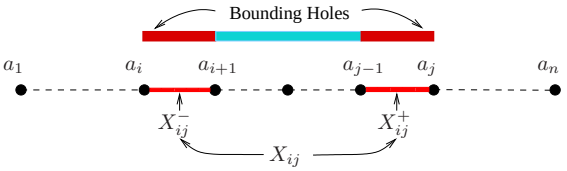


Figure 1: Bounding holes and the partition of X_{ij} into X_{ij}^- and X_{ij}^-

that for any $x \in X^\circ$ the bounding holes of $H(x) = H_{ij}$ belong to $AH(x)$. Otherwise, if say $H_i \notin AH(x)$, then $a_{i+1} \prec x$ in any ϵ-compatible total order $\prec$ extending $\preccurlyeq$, thus we can augment the canonical partial order $\preccurlyeq$ by setting $a_{i+1} \preccurlyeq x$ and by reducing the segments $H(x)$ accordingly. Next we investigate the pairwise admissible locations of x and y in function of the mutual geometric location of the segments $H(x)$ and $H(y)$ and of the values $d(x,y), d_x$, and d_y. We distinguish the following cases: **(H1)** $H(x) = H(y)$; **(H2)** $H(x)$ and $H(y)$ are disjoint; **(H3)** $H(x)$ and $H(y)$ overlap in at least 2 holes ($H(x) \circ H(y)$); **(H4)** $H(x)$ and $H(y)$ overlap in a single hole ($H(x) * H(y)$); **(H5)** $H(y)$ is a proper subinterval of $H(x)$ ($H(y) \Subset H(x)$). This classification of pairs $\{x,y\}$ of X° is used in the design of our approximation algorithm. Also the proofs of several results employ a case analysis based on (H1)-(H5). We continue with the following result. It specifies the constraints on pairs of elements, each element of X° can be located in one of its bounding holes.

Proposition 3.2. *For two elements $x, y \in X^\circ$, any location of x in a bounding hole of $H(x) = H_{ij}$ and any location of y in a bounding hole of $H(y) = H_{i'j'}$ is $(x,y,12)$-admissible, unless $H(x) = H(y)$ and $d(x,y) \ll_3 \max\{d_x, d_y\}$ or $d(x,y) \gg_3 \max\{d_x, d_y\}$, subject to the following three constraints:* (i) *if $H(x) \Subset H(y)$, x and y are located in a common bounding hole, then x is between y and a_{i+1};* (ii) *if $H(x) * H(y)$, then $i < i'$ implies $x \prec y$;* (iii) *if $H(x) = H(y)$, x and y are located in the same bounding hole, and $d_y \ll_4 d_x$, then y is between x and a_{i+1}. If $H(x) = H(y)$ and $d(x,y) \gg_3 \max\{d_x, d_y\}$, then the only $(x,y,1)$-admissible locations are the two locations of x and y in different bounding holes. If $H(x) = H(y)$ and $d(x,y) \ll_3 \max\{d_x, d_y\}$, then any $(x,y,1)$-admissible location is in common x- and y-admissible holes.*

4. Distributing elements to holes

In this section, we describe how, for each hole H_i, to compute the set X_i of elements of X° which will be located in H_i. This set consists of some x such that H_i is a bounding hole of $H(x)$. Additionally, each X_i will be partitioned into an ordered list of cells, to which we perform recursive calls. Let X_{ij} consist of all $x \in X^\circ$ such that $H(x) = H_{ij}$. The sets X_{ij} form a partition of X°. In the next subsections, we will show how to partition each X_{ij} into two subsets X_{ij}^- and X_{ij}^+, so that X_{ij}^- will be located in H_i and X_{ij}^+ in H_{j-1}; see Fig. 1.

4.1. Blocks, cells, and clusters. Two elements $x, y \in X_{ij}$ are called *linked* (*separated*) if in all $(x,y,1)$-admissible locations x and y must be placed in the same hole (in distinct bounding holes). Two subsets A and B of X_{ij} must be *separated* if all $x \in A$ and $y \in B$ are separated. Let S_{ij} and L_{ij} be the sets of all pairs $x, y \in X_{ij}$ such that $d(x,y) \gg_3 \max\{d_x, d_y\}$, resp., $d(x,y) \ll_3 \max\{d_x, d_y\}$. By Proposition 3.2, all pairs of S_{ij} are separated and all pairs of L_{ij} are linked. Since "be linked" is an equivalence relation, all vertices of the same connected component (called *block*) of the graph $\mathcal{L}_{ij} = (X_{ij}, L_{ij})$ are linked. We continue by investigating in which cases two blocks of $\mathcal{L}_{ij}$ are separated or linked. For $x, y \in X_{ij}$, set $x \rightarrowtail y$ iff **(A1)** $d_x \ll_4 d_y$ or **(A2)** $d_x \gtrsim_4 d_y$ and there exists $z \in X_{ij}$ such

that $xz, yz \notin L_{ij}$ and $d(x,z) \ll_{16} d(y,z)$. If $x,y,z \in X_{ij}$ satisfy **(A2)**, then it can be shown that y and z are *strongly separated*, i.e., $d(y,z) \gg_9 \max\{d_y, d_z\}$. Additionally, we show that if $x \rightarrowtail y$, then $x \prec y$ in all ϵ-compatible orders $\prec$ such that $a_{i+1} \prec \{x,y\}$ and $y \prec x$ in all ϵ-compatible orders $\prec$ such that $\{x,y\} \prec a_{j-1}$.

On X_{ij} we define a directed graph $\mathcal{L}_{ij}^{\rightarrow}$: we draw an arc $x \to y$ iff **(L1)** $x \rightarrowtail y$ and x,y belong to a common block of $\mathcal{L}_{ij}$ or **(L2)** $d(x,y) \ll_5 \max\{d_x, d_y\}$. If **(L2)** is satisfied, then $xy \in L_{ij}$ and $y \to x$ hold. The strongly connected components of $\mathcal{L}_{ij}^{\rightarrow}$ are called *cells*. Every block is a disjoint union of cells. Indeed, if x,y belong to a common cell, let R be a directed path of $\mathcal{L}_{ij}^{\rightarrow}$ from x to y. Pick any arc $u \to v$ of R. If it has type **(L2)**, then $uv \in \mathcal{L}_{ij}$. Otherwise, if $u \to v$ has type **(L1)**, then u and v belong to a common block. Thus the ends of all arcs of any path between x,y belong to a common block.

Lemma 4.1. *Let $x,x',y \in X_{ij}$. If x,x' belong to a common cell, but $\{x,x'\}$ and y belong to distinct blocks, then there does not exist an ϵ-compatible order such that $x \prec y \prec x'$.*

Lemma 4.2. *For cells C',C'', if $x,x' \in C'$, $y,y' \in C''$, and $x \rightarrowtail y$, $y' \rightarrowtail x'$, then C' and C'' must be separated.*

Proof. Let B',B'' be the blocks containing C',C''. If $B' = B''$, as $x \rightarrowtail y$ and $y' \rightarrowtail x'$, they are (L1)-arcs, hence $x \to y$ and $y' \to x'$. This is impossible since $\{x,x'\}$ and $\{y,y'\}$ belong to distinct cells. Thus $B' \neq B''$. By Lemma 4.1, if we locate x,x',y,y' in the same bounding hole H_j, either $\{x,x'\} \prec \{y,y'\}$ or $\{y,y'\} \prec \{x,x'\}$ holds. On the other hand, $x \rightarrowtail y$, $y' \rightarrowtail x'$ imply that $x \prec y$ and $y' \prec x'$. Thus C' and C'' must be separated. ∎

Now, let $\mathcal{S}_{ij}$ be a graph having cells as vertices and an edge between two cells C',C'' iff **(S1)** there exist $x,y \in X_{ij}$, x in the same block as C' and y in the same block as C'' such that $xy \in S_{ij}$ or **(S2)** there exist x,x' in the same block as C' and y,y' in the same block as C'' such that each pair xx' and yy' belong to a common cell, and $x \rightarrowtail y, y' \rightarrowtail x'$. By Proposition 3.2 and Lemma 4.2, in cases **(S1)** and **(S2)** the sets C' and C'' must be separated. The graph $\mathcal{S}_{ij}$ must be bipartite, otherwise no ϵ-compatible order exist. Now, for each connected component of $\mathcal{S}_{ij}$ consider its canonical bipartition $\{A',A''\}$, and draw an edge between any two cells, one from A' and another from A''. Denote the obtained graph also by $\mathcal{S}_{ij}$. Call the union of cells from A' (or from A'') a *cluster*. The clusters $\mathcal{K}'$ and $\mathcal{K}''$ of A' and A'' are called *twins*. From the construction, we immediately obtain that all elements of a cluster are linked and two twin clusters are separated. A connected bipartite component $\{\mathcal{K}',\mathcal{K}''\}$ of $\mathcal{S}_{ij}$ is called a *principal component* if there exists $x \in \mathcal{K}'$ and $y \in \mathcal{K}''$ such that x and y are strongly separated.

4.2. Partitioning X_{ij} into X_{ij}^- and X_{ij}^+. We describe how to partition X_{ij} into the subsets X_{ij}^- and X_{ij}^+. For this, we define a directed graph $\mathcal{G}_{ij}$ having cells as vertices, and an arc $C' \rightarrowtail C$ with tail C' and head C exists iff one of the following conditions is satisfied: **(G1)** C' and C belong to twin clusters of $\mathcal{S}_{ij}$; **(G2)** C' and C are not connected by (G1)-arcs and there exist $x \in C$ and $x' \in C'$ such that $d_{x'} \ll_4 d_x$; **(G3)** C' and C are not connected by (G1)- or (G2)-arcs and there exist $x \in C, x' \in C'$, and $z \in X_{ij}$ such that $xz, x'z \notin L_{ij}$ and $d(x',z) \ll_{16} d(x,z)$. A head of a (G3)-arc is called a (G3)-*cell*. A (Gi)-*cycle* is a directed cycle of $\mathcal{G}_{ij}$ with arcs of type **(Gi)**, $i = 1,2,3$. The (G1)-cycles are exactly the cycles of length 2. A *mixed cycle* is a directed cycle containing arcs of types **(G2)** and **(G3)**. Finally, an *induced cycle* is a directed cycle $\mathcal{C}$ such that for two cells $C,C' \in \mathcal{C}$ we have $C' \rightarrowtail C$ if and only if C is the successor of C' in $\mathcal{C}$. Our next goal is to establish that either the set

of cells can be partitioned into two subsets such that the subgraphs of $\mathcal{G}_{ij}$ induced by these subsets do not contain directed cycles or no ϵ-compatible order exist. Deciding if a directed graph can be partitioned into two acyclic subgraphs is NP-complete [17]. In our case, this can be done in polynomial time by exploiting the structure of $\mathcal{G}_{ij}$.

Lemma 4.3. *If $\mathcal{C} = (C_1, C_2, \ldots, C_k, C_1)$ is a directed cycle of $\mathcal{G}_{ij}$, then for any ϵ-compatible order, $\mathcal{C}$ has a cell located in the hole H_i and a cell located in the hole H_{j-1}.*

Proof. The assertion is obvious if $\mathcal{C}$ is a (G1)-cycle. So, suppose that all arcs of $\mathcal{C}$ have type **(G2)** or **(G3)**. The definition of cells implies that $\mathcal{C}$ contains two consecutive cells, say C_1 and C_k, which belong to different blocks. Suppose that there exists an ϵ-compatible order $\prec$ such that no element of $\cup_{l=1}^{k} C_l$ is located in the hole $H_i = [a_i, a_{i+1}]$, i.e., $a_{i+1} \prec \cup_{l=1}^{k} C_l$. In each C_l pick two elements x_l, y_l such that $x_l \rightarrowtail y_{l+1(modk)}$. Then $x_l \prec y_{l+1(modk)}$ for all $l = 1, \ldots, k$. We divide the cells of $\mathcal{C}$ into groups: a group consists of all consecutive cells of $\mathcal{C}$ belonging to one and the same block. The first group starts with C_1, while the last group ends with C_k. We assert that if $\{C_{l-q}, \ldots, C_l\}$ and $\{C_{l+1}, \ldots, C_{l+r}\}$ are two consecutive groups of $\mathcal{C}$, then $C_l \prec C_{l+1} \cup \cdots \cup C_{l+r}$ (all indices here are modulo k). Indeed, pick $u \in C_l$ and $v \in C_{l+1}$. Since $\{x_l, u\}$ and $\{y_{l+1}, v\}$ belong to different blocks while each of these pairs belong to a common cell, applying Lemma 4.1 to each of the triplets of the quadruplet x_l, u, y_{l+1}, v, we infer that in the total order $\prec$ none of y_{l+1}, v is located between x_l and u and none of x_l, u is located between y_{l+1} and v. Since $x_l \prec y_{l+1}$, we conclude that $\{x_l, u\} \prec \{y_{l+1}, v\}$, yielding $C_l \prec C_{l+1}$. Now, consider the cell C_{l+2}. The element y_{l+2} must be located to the right of x_{l+1}, therefore to the right of C_l. Since C_{l+2} and C_l belong to different blocks, we can show that $C_l \prec C_{l+2}$ by using exactly the same reasoning as for the cells C_l and C_{l+1}. Continuing this way, we obtain the required relationship $C_l \prec C_{l+1} \cup \cdots \cup C_{l+r}$. This establishes the assertion. Suppose that $[1, i_1], [i_1 + 1, i_2], \ldots, [i_r + 1, k]$ are the indices of cells defining the beginning and the end of each group. From our assertion we infer that $C_k \prec C_{i_1} \prec C_{i_2} \prec \ldots \prec C_{i_r} \prec C_k$, contrary that $\prec$ is a total order. ∎

Lemma 4.4. *If $C \rightarrowtail C'$ is a (G3)-arc and C belongs to a principal component, then C and C' belong to the same cluster. In particular, $\mathcal{G}_{ij}$ does not contain (G3)-cycles or no ϵ-compatible order exist. Moreover, $\mathcal{G}_{ij}$ does not contain (G2)-cycles.*

Proof. Let xy be a strongly separated pair with $x \in C$. Since $C \rightarrowtail C'$ is a (G3)-arc, there exist $y' \in C$ and $x' \in C'$ such that $y' \rightarrowtail x'$ is an (A2)-arc. Then there exists z' such that $x'z'$ is strongly separated. If xz and $x'y'$ belong to different principal components, then there exists a (G2)-arc from C' to C or from C to C'. In the first case, C and C' obey **(S2)**, thus we cannot have a (G3)-arc from C to C'. Analogously, in the second case, we deduce that we have at the same time a (G3)-arc and a (G2)-arc from C to C'. This is impossible, so C and C' belong to a common principal component. Now, if $\mathcal{G}_{ij}$ contains a (G3)-cycle, then the first assertion implies that all its cells belong to the same cluster, and Lemma 4.3 yields that no ϵ-compatible order exist. Finally, let $\mathcal{C} = (C_1, C_2, \ldots, C_k, C_1)$ be a (G2)-cycle. In each C_i, pick x_i, y_i so that $d_{x_i} \ll_4 d_{y_{i+1(modk)}}$. Since there is no **(G2)** or **(G3)** arc from $C_{i+1(modk)}$ to C_i, we get $d_{y_i} \lesssim_4 d_{x_{i+1(modk)}}$, yielding $d_{x_i} \ll_4 d_{y_{i+1(modk)}} \lesssim_4 d_{x_{i+2(modk)}}$. Thus $d_{x_i} < d_{x_{i+2(modk)}}$ for $i = 1, \ldots k$. Then $d_{x_1} < d_{x_3} < \cdots < d_{x_{k-1}} < d_{x_1}$ for even k and $d_{x_1} < d_{x_3} < \cdots < d_{x_k} < d_{x_2} < d_{x_4} < \cdots < d_{x_{k-1}} < d_{x_1}$ for odd k, a contradiction. ∎

To complete the bipartition of cells into two acyclic subgraphs of $\mathcal{G}_{ij}$, it remains to deal with induced mixed cycles. The following results precise their structure.

Lemma 4.5. *Any induced mixed cycle $\mathcal{C}$ of $\mathcal{G}_{ij}$ contains one or two (G2)-arcs, and if $\mathcal{C}$ contains two such arcs, then they are consecutive.*

Lemma 4.6. *Let $C' \rightarrowtail C$ be a (G3)-arc, $C \rightarrowtail C''$ be a (G2)-arc, and suppose that there is no (G2)-arc from C' to C''. If C, C' do not belong to distinct twin clusters and C, C'' do not belong to the same cluster, then C and C' must be separated.*

Thus a mixed cycle $\mathcal{C}$ contains either one (G2)-arc ($\mathcal{C}$ is a 1-*cycle*) or two consecutive (G2)-arcs ($\mathcal{C}$ is a 2-*cycle*), all other arcs of $\mathcal{C}$ being (G3)-arcs. By Lemma 4.4, the heads of all (G3)-arcs of $\mathcal{C}$ are (G3)-cells of the same cluster $\mathcal{K}$. Then we say that the cycle $\mathcal{C}$ *intersects* the cluster $\mathcal{K}$. For a (G2)-arc $C_0 \rightarrowtail C$ and a cluster $\mathcal{K}$, we show how to detect if there exists a 1- or 2-cycle $\mathcal{C}$ passing via $C_0 \rightarrowtail C$ and intersecting $\mathcal{K}$. We consider the case of 1-cycles. Then C_0 must be a (G3)-cell of $\mathcal{K}$. Note that an induced 1-cycle cannot contain cells C' such that $C_0 \rightarrowtail C'$ is a **(G2)** or **(G3)**-arc. Hence, we can remove all such cells of $\mathcal{K}$. Analogously, we remove all cells C' so that $C' \rightarrowtail C$ is an arc. In the subgraph induced by the remaining cells of $\mathcal{K}$ we search for a shortest directed path $\mathcal{Q} = C \rightarrowtail C_1 \rightarrowtail \cdots \rightarrowtail C_k \rightarrowtail C_0$ so that the first arc $C \rightarrowtail C_1$ and the last arc $C_k \rightarrowtail C_0$ of this path are (G3)-arcs. This can be done in polynomial time by testing all possible choices for C_1 and C_k and applying for each pair a shortest path finding algorithm in an acyclic graph. If such a path $\mathcal{Q}$ does not exist, then no required induced cycle $\mathcal{C}$ exist. Otherwise, the path $\mathcal{Q}$ together with the arc $C_0 \rightarrowtail C$ define an induced cycle $\mathcal{C}$ having exactly one (G2)-arc. Indeed, if $C_i \rightarrowtail C_j$ is a **(G2)** or **(G3)**-arc and $|i-j| > 2$, since the subgraph induced by $\mathcal{K}$ is acyclic, we must have $i < j$. This contradicts the minimality of the path $\mathcal{Q}$. So, the resulting cycle is indeed induced. It remains to note that $\mathcal{C}$ does not contain other (G2)-arcs, because by Lemma 4.5 in an induced cycle the (G2)-arcs are consecutive. Analogously, we can decide if there exists a 2-cycle passing via $C_0 \rightarrowtail C$ and intersecting $\mathcal{K}$, and having a second (G2)-arc of the form $C \rightarrowtail C_0'$ or $C_0' \rightarrowtail C_0$. Therefore, we have the following result:

Lemma 4.7. *For a (G2)-arc $C_0 \rightarrowtail C$ and a cluster $\mathcal{K}$, one can decide in polynomial time if there exists an induced 1- or 2-cycle $\mathcal{C}$ passing via $C' \rightarrowtail C$ and intersecting $\mathcal{K}$.*

For a cell C, let $\Omega_1(C)$ be the set of (G2)-arcs $C_0 \rightarrowtail C$ belonging to a 1-cycle intersecting a cluster $\mathcal{K}$ not containing C. Let $\Omega_2(C)$ be the set of (G2)-arcs $C_0 \rightarrowtail C$ belonging to a 2-cycle $\mathcal{C}$ intersecting a cluster $\mathcal{K}$ not containing C and passing via $C_0 \rightarrowtail C$ so that the arc of $\mathcal{C}$ entering C_0 is a (G3)-arc. In both cases C_0 belongs to $\mathcal{K}$: C_0 is a head of a (G3)-arc of $\mathcal{C}$, and all such heads belong to $\mathcal{K}$. Finally, let $\Omega_3(C)$ be the set of (G2)-arcs $C \rightarrowtail C_0$ belonging to a 2-cycle $\mathcal{C}$ intersecting a cluster $\mathcal{K}$, so that C belongs to $\mathcal{K}$ and the arc of $\mathcal{C}$ entering C has type **(G2)**. Fig. 2 illustrates this classification. For each cell C of $\mathcal{G}_{ij}$ we introduce a binary variable x_C satisfying the following constraints: **(F1)** $x_{C'} = x_{C''}$, if C', C'' belongs to the same cluster; **(F2)** $x_{C'} \neq x_{C''}$, if C', C'' belong to twin clusters; **(F3)** $x_C \neq x_{C_0}$, if the arc $C_0 \rightarrowtail C$ belongs to $\Omega_1(C) \cup \Omega_2(C)$; **(F4)** $x_C \neq x_{C_0}$, if the arc $C \rightarrowtail C_0$ belongs to $\Omega_3(C)$. Define a 2-SAT formula Φ_{ij} by replacing every constraint $a = b$ by two clauses $(a \vee \bar{b})$ and $(\bar{a} \vee b)$ and every constraint $a \neq b$ by two clauses $(a \vee b)$ and $(\bar{a} \vee \bar{b})$.

Proposition 4.8. *If the 2-SAT formula Φ_{ij} admits a satisfying assignment A, then the sets $X_{ij}^- = \{C : A(x_C) = 0\}$ and $X_{ij}^+ = \{C : A(x_C) = 1\}$ define a partition of $\mathcal{G}_{ij}$ into two acyclic subgraphs. Conversely, given an ϵ-compatible order on X, the assignment A defined by setting $A(x_C) = 0$ if C is located in H_i, $A(x_C) = 1$ if C is located in H_{j-1}, and $A(x_{C'}) = A(x_{C''})$ if C' and C'' are located in a common inner hole, is a true assignment for Φ_{ij}. In particular, if Φ_{ij} is not satisfiable, then no ϵ-compatible order exist.*

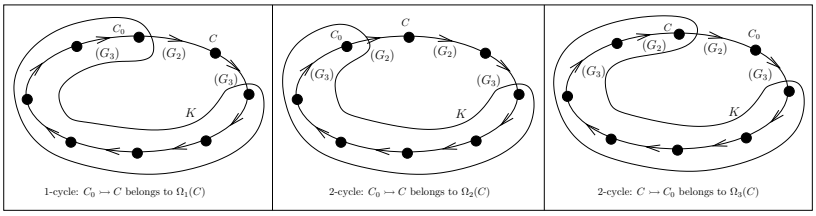


Figure 2: To the classification of the arcs incident to a cell C

Proof. Let A be a true assignment of Φ_{ij} and the partition X_{ij}^-, X_{ij}^+ of X_{ij} be defined as above. Denote by $\mathcal{G}_{ij}^-$ and $\mathcal{G}_{ij}^+$ the subgraphs induced by X_{ij}^- and X_{ij}^+. **(F1)** forces every cluster to be included in one set. **(F2)** implies that the twin clusters are separated. Hence $\mathcal{G}_{ij}^-$ and $\mathcal{G}_{ij}^+$ do not contain (G1)-cycles: if C and C' are the two cells of a (G1)-cycle, then $(x_C \vee x_{C'}) \wedge (\bar{x}_C \vee \bar{x}_{C'})$ yields $A(x_C) \neq A(x_{C'})$. By Lemma 4.4, $\mathcal{G}_{ij}$ does not contain (G2)-cycles. Since the cells of a (G3)-cycle are contained in the same cluster and each cluster induces an acyclic subgraph, $\mathcal{G}_{ij}^-$ and $\mathcal{G}_{ij}^+$ do not contain (G3)-cycles as well. Now, let $\mathcal{G}_{ij}^+$ contain a mixed cycle. Then it also contains an induced mixed cycle $\mathcal{C}$. From Lemma 4.5 we infer that $\mathcal{C}$ has either one (G2)-arc $C_0 \rightarrowtail C$ or exactly two consecutive (G2)-arcs $C_0 \rightarrowtail C \rightarrowtail C''$. In the first case, we conclude that $C_0 \rightarrowtail C$ belongs to $\Omega_1(C)$, thus **(F3)** yields $x_C \neq x_{C_0}$, contrary to the fact that $A(x_C) = A(x_{C'}) = 1$. Analogously, in the second case, we deduce that either $x_C \neq x_{C_0}$ and the arc $C_0 \rightarrowtail C$ belongs to $\Omega_2(C)$ or $x_C = x_{C_0}$ and the arc $C \rightarrowtail C''$ belongs to $\Omega_3(C)$, whence $x_C \neq x_{C''}$. Then we obtain a contradiction with the assumption that $A(x_{C_0}) = A(x_C) = A(x_{C''}) = 1$. This shows that the subgraphs $\mathcal{G}_{ij}^-$ and $\mathcal{G}_{ij}^+$ obtained from the true assignment A of Φ_{ij} are acyclic.

Conversely, let A be an assignment obtained from an ϵ-compatible order as defined in the proposition. We assert that A is a true assignment for Φ_{ij}, i.e., it satisfies the constraints **(F1)-(F4)**. This is obvious for constraints **(F1)** and **(F2)**, because if two cells C', C'' belong to the same cluster, then they will be located in the same hole and we must have $A(x_{C'}) = A(x_{C''})$. If C' and C'' belong to distinct twin clusters, then they must be separated, therefore the unique ϵ-admissible location of C' and C'' will be in different bounding holes, thus $A(x_{C'}) \neq A(x_{C''})$. Now, pick an arc $C_0 \rightarrowtail C$ which belongs to $\Omega_1(C) \cup \Omega_2(C)$. If $C_0 \rightarrowtail C$ belongs to $\Omega_1(C)$, then there exists a 1-cycle $\mathcal{C}$ passing via $C_0 \rightarrowtail C$ and intersecting a cluster $\mathcal{K}$. Since all cells of $\mathcal{C}$, except C, are heads of (G3)-arcs, they all belong to $\mathcal{K}$, i.e., they have the same value in the assignment. By Lemma 4.3, C must be separated from C_0 (namely C and C' must be located in different bounding holes), showing that $A(x_C) \neq A(x_{C_0})$. If $C_0 \rightarrowtail C$ belongs to $\Omega_2(C)$, then let $\mathcal{C}$ be a 2-cycle passing via $C_0 \rightarrowtail C$ and intersecting the cluster $\mathcal{K}$ not containing C. Additionally, we know that the arc $C' \rightarrowtail C_0$ of $\mathcal{C}$ entering C_0 is a (G3)-arc, thus C_0 belongs to $\mathcal{K}$. Since C' cannot belong to the twin cluster of $\mathcal{K}$ (this will contradicts that $C' \rightarrowtail C_0$ is a (G3)-arc) and since C does not belong to $\mathcal{K}$, from Lemma 4.6 we infer that C_0 and C are separated, thus $A(x_C) \neq A(x_{C_0})$. Finally, let $C \rightarrowtail C_0$ belong to $\Omega_3(C)$. Then there exists a 2-cycle $\mathcal{C}$ passing via $C \rightarrowtail C_0$ and intersecting the cluster $\mathcal{K}$, such that C belongs to $\mathcal{K}$ and the arc of $\mathcal{C}$ entering C has type **(G2)**. Since all cells of $\mathcal{C}$ except C and C_0 are heads of (G3)-arcs, they all belong to $\mathcal{K}$. Since C also belongs to this cluster, by Lemma 4.3, C_0 must be separated from the remaining cells of $\mathcal{C}$, yielding $x_C \neq x_{C_0}$. Hence A satisfies the constraints **(F1)-(F4)**. This shows, in particular, that if Φ_{ij} is not satisfiable, then no ϵ-compatible order exist. ∎

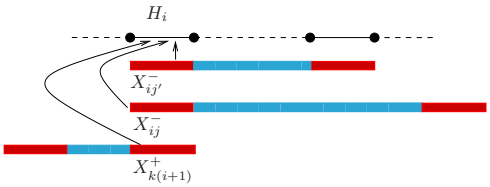


Figure 3: Relative location of the cells of $X^+_{k(i+1)}, X^-_{ij'}$, and X^-_{ij} $(k < i, j' < j)$ in H_i

4.3. Sorting the cells of X^-_{ij} and X^+_{ij}. Let $\mathcal{G}^-_{ij}$ and $\mathcal{G}^+_{ij}$ be the subgraphs of $\mathcal{G}_{ij}$ induced by the sets X^-_{ij} and X^+_{ij} obtained from the true assignment of the 2-SAT formula Φ_{ij}. We will locate all cells of X^-_{ij} in the hole H_i and all cells of X^+_{ij} in the hole H_{j-1} of H_{ij}. The elements from two cells C', C'' located in the same hole will not be mixed, i.e., C' will be placed to the right of C'', or vice versa. To specify the total order among cells, we use that $\mathcal{G}^-_{ij}$ and $\mathcal{G}^+_{ij}$ are acyclic, therefore each of them admit a topological order. We compute a topological order $C_{j_1} \prec C_{j_2} \prec \ldots \prec C_{j_p}$ on the cells of X^+_{ij} and a dual topological order $C_{i_q} \prec C_{i_{q-1}} \prec \ldots \prec C_{i_1}$ on the cells of X^-_{ij}. We locate the cells of X^+_{ij} in H_{j-1} and the cells of X^-_{ij} in H_i according to these orders. The following two results relay the topological orders on the cells with the order on the distances between elements from such cells.

Lemma 4.9. *Let C', C'' be two cells of X^+_{ij}. If $C' \prec C''$ in the topological order, then for any $y \in C'$, $z \in C''$ and $x \in X^-_{ij}$, we have $d_y \lesssim_4 d_z$ and $d(x,y) \lesssim_{16} d(x,z)$.*

Proof. Since C', C'' belong to X^+_{ij}, they are not connected by (G1)-arcs. Since $C' \prec C''$ in the topological order, there is no arc from C'' to C'. As $C'' \rightarrowtail C'$ is not a (G2)-arc, we must have $d_z \gtrsim_4 d_y$. As $C'' \rightarrowtail C'$ is not a (G3)-arc, we obtain $d(x,y) \lesssim_{16} d(x,z)$. ∎

Lemma 4.10. *Let C, C', C'' be three distinct cells of the graph $\mathcal{G}_{ij}$. If the algorithm returns the total order $\prec$ and $C \prec C' \prec C''$, then for any $x \in C, y \in C', z \in C''$ or $x, y, z \in C \cup C'$ and $x \prec y \prec z$, we have $d(x,z) \gtrsim_{16} \max\{d(x,y), d(y,z)\}$.*

After fixing the relative position of each cell C of X_{ij}, we make a recursive call to C. For this, we update the canonical order $\preccurlyeq$ in the following way: if C is located in X^+_{ij}, we set $x \preccurlyeq^+ y$ if $x \rightarrowtail y$, otherwise, if C is located in X^-_{ij}, we set $x \preccurlyeq^- y$ if $y \rightarrowtail x$. Since $\preccurlyeq^+$ and $\preccurlyeq^-$ are dual, if we apply to them the "closing" rules, we will obtain two dual partial orders, denoted also by $\preccurlyeq^+$ and $\preccurlyeq^-$. The restriction on C of every ϵ-compatible order $\prec$ on X is an extension of $\preccurlyeq^+$ or $\preccurlyeq^-$: since all elements of C will be placed in the same hole, either $a_{i+1} \prec C$ or $C \prec a_j$. If $a_{i+1} \prec C$, then $x \prec y$ for all $x, y \in C$ such that $x \rightarrowtail y$. Hence $\prec$ is a linear extension of $\preccurlyeq^+$. Therefore, if the recursive call to a cell C returns the answer "not", then no ϵ-compatible total order on X exist. Else, it returns a total order on C, which is 16ϵ-compatible by induction hypothesis. Then, the total order between the cells of $\mathcal{G}_{ij}$ and the total orders on cells are concatenated to give a single total order $\prec$ on X_{ij}.

4.4. Defining the total order on X_i. Recall that X_i is the set of all elements of X° located in the hole H_i. According to our algorithm, X_i is the disjoint union of all sets X^-_{ij} $(j > i+1)$ and $X^+_{k(i+1)}$ $(k < i)$. We just defined a total order between the cells of each of the sets $X^-_{ij}, X^+_{k(i+1)}$, and applying recursion we defined a total order on the elements of each cell. To obtain a total order on the whole set X_i it remains to define a total order between the sets X^-_{ij} $(j > i+1)$ and $X^+_{k(i+1)}$ $(k < i)$. For this, we locate each $X^+_{k(i+1)}$ $(k < i)$ to the

left of each X^-_{ij} $(j > i)$. Given two sets $X^+_{k(i+1)}, X^+_{k'(i+1)}$ $(k, k' < i)$, we locate $X^+_{k(i+1)}$ to the left of $X^+_{k'(i+1)}$ if and only if $k < k'$, i.e., iff $H_{k(i+1)} \Subset H_{k'(i+1)}$. Analogously, given $X^-_{ij}, X^-_{ij'}$ $(j, j' > i+1)$, we locate $X^-_{ij'}$ to the right of X^-_{ij} if and only if $j' < j$, i.e., iff $H_{ij'} \Subset H_{ij}$. This location is justified by the Proposition 3.2 and is illustrated in Fig. 3.

5. The algorithm and its performance guarantee

We have collected all necessary tools to describe the algorithm. It consists of three procedures l_∞-Fitting_by_Robinson, Refine, and Partition_and_Sort. The main procedure l_∞-Fitting_by_Robinson constructs the sorted list Δ of feasible values for the optimal error ϵ^*. Its entries are considered in a binary search fashion and the algorithm returns the smallest value $\epsilon \in \Delta$ occurring in this search for which the answer "not" is not returned (i.e., the least ϵ for which a 16ϵ-compatible total order on X exists). To decide, if, for a given ϵ, such an order exists, the procedure Refine$(X, \preccurlyeq, \epsilon)$ constructs (and/or updates) the canonical partial order $\preccurlyeq$ and computes a maximal chain P of $(X, \preccurlyeq)$. For each element $x \in X^\circ := X \setminus P$, Refine computes the set $AH(x)$ of all x-holes which participate in $(x, y, 1)$-admissible locations for all $y \in X^\circ$ and defines the segment $H(x)$. For each pair $i < j - 1$, Refine constructs the set X_{ij} and makes a call of the procedure Partition_and_Sort(X_{ij}), which returns the bipartition $\{X^-_{ij}, X^+_{ij}\}$ of X_{ij} and a total order on the cells of X^-_{ij} and X^+_{ij}. Then Refine concatenates in a single total order on cells the total orders on cells coming from different sets assigned to the same hole. After this, Refine is recursively applied to each cell occurring in some graph $\mathcal{G}_{ij}$. The returned total orders on cells are concatenated into a single total order $\prec$ on X according to the total orders between cells and between holes; then $\prec$ is returned by the algorithm l_∞-Fitting_by_Robinson. The procedure Partition_and_Sort constructs the graphs $\mathcal{L}_{ij}$ and $\mathcal{L}^{\rightarrow}_{ij}$. Using these graphs, X_{ij} is partitioned into blocks and cells, then graph $\mathcal{S}_{ij}$ and its clusters are constructed. Using the cells, the directed graph $\mathcal{G}_{ij}$ is constructed. If $\mathcal{S}_{ij}$ is not bipartite or $\mathcal{G}_{ij}$ contains (G3)-cycles, then Partition_and_Sort returns the answer "not". Otherwise, for each cell C and each cluster $\mathcal{K}$, it tests if there exists a 1-cycle and/or a 2-cycle passing via C and intersecting $\mathcal{K}$. Consequently, for each cell C, the lists $\Omega_1(C), \Omega_2(C)$, and $\Omega_3(C)$ of (G2)-arcs are computed. These lists are used to construct the 2-SAT formula Φ_{ij}, which is solved by the algorithm of [2]. If Φ_{ij} admits a true assignment A, then $X^-_{ij} = \{C : A(x_C) = 0\}$ and $X^+_{ij} = \{C : A(x_C) = 1\}$ define a bipartition of X_{ij} into two acyclic subgraphs $\mathcal{G}^-_{ij}, \mathcal{G}^+_{ij}$ of $\mathcal{G}_{ij}$. Then Partition_and_Sort locates the cells from X^+_{ij} in the hole H_{j-1} according to the topological order of the acyclic graph $\mathcal{G}^+_{ij}$ and it locates the cells from X^-_{ij} in the hole H_i according to the dual topological order of $\mathcal{G}^-_{ij}$. Note that if at some stage Refine or Partition_and_Sort returns the answer "not", then there does not exists any ϵ-compatible total order on X and the current value of ϵ is too small. The total complexity of the algorithm is $O(n^6 \log n)$. We formulate now the main result of our paper:

Theorem 5.1. *For $\epsilon \in \Delta$, if the algorithm returns the answer "not", then the dissimilarity d is not ϵ-Robinson, else, it returns a 16ϵ-compatible total order $\prec$ on X. In particular, the algorithm is a factor 16 approximation algorithm for l_∞-*FITTING-BY-ROBINSON.

Proof. First, note that no ϵ-compatible order exist in all cases when the algorithm returns the answer "not". Indeed, Lemma 4.4, Propositions 3.2 and 4.10 cover all such cases except the case when this answer is returned by a recursive call. In this case, the induction

assumption implies that no ϵ-compatible total order on C extending $\preccurlyeq^+$ (and therefore its dual $\preccurlyeq^-$) exist. Then we infer that no ϵ-compatible order on X exist as well.

Now, let the algorithm return a total order $\prec$. Suppose by induction assumption that $\prec$ is 16ϵ-compatible on each cell to which a recursive call is applied. On the chain P, the total order $\prec$ coincides with $\preccurlyeq$, therefore $\prec$ is ϵ-compatible on P. Moreover, $\prec$ is ϵ-compatible on $P \cup \{x\}$ for any $x \in X^\circ$, because every element x is located in a bounding hole of $H(x)$ which is x-admissible. Finally notice that $\prec$ is 12ϵ-compatible on $P \cup \{x, y\}$ for any $x, y \in X^\circ$ because by Proposition 3.2 the bounding hole of $H(x)$ and the bounding hole of $H(y)$ into which x and y are located define a $(x, y, 12)$-admissible pair. To prove that $\prec$ is 16ϵ-compatible on the whole set X, it suffices to show that $d(x, z) \gtrsim_{16} \max\{d(x, y), d(y, z)\}$ for any three elements $x, y, z \in X$ such that $x \prec y \prec z$. From previous discussion, we can suppose that $x, y, z \in X^\circ$. For this, we distinguish the Cases **(H1)-(H5)** in function of the mutual location of segments $H(x)$ and $H(z)$ and in each case we show the required inequality. The respective case analysis is given in [9].

■

References

[1] R. Agarwala, V. Bafna, M. Farach, B. Narayanan, M. Paterson, and M. Thorup, On the approximability of numerical taxonomy (fitting distances by tree metrics), *SIAM J. Comput.* **17** (1999), 1073-1085.

[2] B. Aspvall, M.F. Plass, and R.E. Tarjan, A linear time algorithm for testing the truth of certain quantified boolean formulas, *Inf. Proc. Lett.*, **8** (1979), 121-123.

[3] J.E. Atkins, E.G. Boman, and B. Hendrickson, A spectral algorithm for seriation and the consecutive ones problem, *SIAM J. Comput.* **28** (1998), 297–310.

[4] M. Bădoiu, Approximation algorithm for embedding metrics into a two-dimensional space, *SODA 2003.*

[5] M. Bădoiu, A. Gupta, K. Dhamdhere, Y. Rabinovich, H. Räcke, and R. Ravi, A. Sidiropoulos, Approximation algorithms for low-distortion embeddings into low-dimensional spaces, *SODA, 2005.*

[6] G. Caraux and S. Pinloche, PermutMatrix: a graphical environment to arrange gene expression profiles in optimal linear order, *Bioinformatics* **21**(2005), 1280-1281.

[7] V. Chepoi and B. Fichet, l_∞-Approximation via subdominants, *J. Math. Psych.,* **44** (2000), 600-616.

[8] V. Chepoi, B. Fichet, and M. Seston, Seriation in the presence of errors: NP-hardness of l_∞-fitting Robinson structures to dissimilarity matrices, *J. Classification* (to appear).

[9] V. Chepoi and M. Seston, Seriation in the presence of errors: an approximation algorithm for fitting Robinson structures to dissimilarity matrices (submitted).

[10] B. Chor and M. Sudan, A geometric approach to betweenness, *SIAM J. Discr. Math.* **11** (1998), 511–523.

[11] E. Diday, Orders and overlapping clusters by pyramids, In *Multidim. Data Analysis,* pp. 201-234, 1986.

[12] C. Durand and B. Fichet, One-to-one correspondences in pyramidal representation: a unified approach, In *Classification and Related Methods of Data Analysis,* pp. 85-90, North–Holland, 1988.

[13] M. Farach, S. Kannan, et T. Warnow, A robust model for finding optimal evolutionary trees, *Algorithmica,* **13** (1995), 155-179.

[14] M. Hahsler, K. Hornik, and C. Buchta, Getting things in order: an introduction to the R package seriation, *J. Statistical Software,* **25** (2008), 1-34.

[15] J. Hstad, L. Ivansson, and J. Lagergren, Fitting points on the real line and its application to RH mapping, In *ESA 1998* and *J. Algorithms,* **49** (2003), 42-62.

[16] L.J. Hubert, Some applications of graph theory and related nonmetric techniques to problems of approximate seriation, *British J. Math. Stat. Psych.*, **27** (1974), 133-153.

[17] D.S. Johnson, The NP-completeness column: an outgoing guide, *J. Algorithms,* **3**(1982), 182-195.

[18] D.G. Kendall, Seriation from abundance matrices, *Mathematics in the Archaeological and Historical Sciences,* Eds., F. R. Hodson, D. G. Kendall, and P. Tăutu, pp. 215-252, 1971.

[19] B. Mirkin and S. Rodin, *Graphs and Genes*, Springer, 1984.

[20] W. S. Robinson, A method for chronologically ordering archaeological deposits, *American Antiquity,* **16** (1951), 293–301.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 277–288
www.stacs-conf.org

ALMOST-UNIFORM SAMPLING OF POINTS ON HIGH-DIMENSIONAL ALGEBRAIC VARIETIES

MAHDI CHERAGHCHI [1] AND AMIN SHOKROLLAHI [1]

[1] EPFL, Switzerland
E-mail address: {mahdi.cheraghchi,amin.shokrollahi}@epfl.ch

Abstract. We consider the problem of uniform sampling of points on an algebraic variety. Specifically, we develop a randomized algorithm that, given a small set of multivariate polynomials over a sufficiently large finite field, produces a common zero of the polynomials almost uniformly at random. The statistical distance between the output distribution of the algorithm and the uniform distribution on the set of common zeros is polynomially small in the field size, and the running time of the algorithm is polynomial in the description of the polynomials and their degrees provided that the number of the polynomials is a constant.

1. Introduction

A natural and important class of problems in computer science deals with random generation of objects satisfying certain properties. More precisely, one is interested in an efficient algorithm that, given a *compact* description of a set of objects, outputs an element in the set uniformly at random, where the exact meaning of "compact" depends on the specific problem in question.

Uniform sampling typically arises for problems in NP. Namely, given an instance belonging to a language in NP, one aims to produce a witness uniformly at random. Here, the requirement is stronger than that of decision and search problems. In a seminal paper, Jerrum, Valiant and Vazirani [8] gave a unified framework for this problem and showed that, for polynomial-time verifiable relations xRy, uniform sampling of a witness y for a given instance x is reducible to approximate counting of the witnesses, and hence, can be efficiently accomplished using a Σ_2^{P} oracle. It is natural to ask whether the requirement for an Σ_2^{P} oracle can be lifted. In fact, this is the case; a result of Bellare, Goldreich, and Petrank [3] shows that an NP oracle is sufficient and also necessary for uniform sampling of NP witnesses.

The NP sampling problem can be equivalently stated as follows: Given a boolean circuit of polynomially bounded size, sample an input that produces the output 1 (if possible), uniformly at random among all possibilities. This problem can be naturally generalized to

Key words and phrases: Uniform Sampling, Algebraic Varieties, Randomized Algorithms, Computational Complexity.

Research supported by Swiss NSF grant 200020-115983/1.

models of computation other than small boolean circuits, and an interesting question to ask is the following: For what restricted models, the uniform (or almost-uniform) sampling problem is efficiently solvable (e.g., by polynomial-time algorithms or polynomial-sized circuits) without the need for an additional oracle? Of course if the role of the NP oracle in [3] can be replaced by a weaker oracle that can be efficiently implemented, that would immediately imply an efficient uniform sampler. While for general NP relations the full power of an NP oracle is necessary, this might not be the case for more restricted models.

In this work, we study the sampling problem for the restricted model of *polynomial functions*. A polynomial function of degree d over a field $\mathbb{F}$ (that we assume to be finite) is a mapping $f\colon \mathbb{F}^n \to \mathbb{F}^m$ such that every coordinate of the output can be computed by an n-variate polynomial of total degree at most d over $\mathbb{F}$. The corresponding sampling problem (that we call *variety sampling*) is defined as follows: Given a polynomial function, find a pre-image of a given output (that can be considered the zero vector without loss of generality) uniformly at random. Hence, in this problem one seeks to sample a uniformly random point on a given algebraic variety. It is not difficult to show that this problem is, in general, NP-hard. Hence, it is inevitable to relax the generality of the problem if one hopes to obtain an efficient solution without the need for an NP oracle. Accordingly, we restrict ourselves to the case where

(1) The co-dimension of the variety (or, the number of the polynomials that define the variety) is *small*,
(2) The underlying field is *sufficiently large*,
(3) The output distribution is only required to be *statistically close* to the uniform distribution on the variety.

It is shown in [8] that almost uniform generation of NP witnesses (with respect to the statistical distance) is possible without using an NP oracle for self-reducible relations for which the size of the solution space can be efficiently approximated. The relation underlying the variety sampling problem consists of a set of n-variate polynomials over $\mathbb{F}$ and a point $x \in \mathbb{F}^n$, and it holds if and only if x is a common zero of the polynomials. Obviously, assuming that field operations can be implemented in polynomial time, this is a polynomial-time verifiable relation. Moreover, the relation is self reducible, as any fixing of one of the coordinates of the witness x leads to a smaller instance of the problem itself, defined over $n-1$ variables. Approximate counting of the witnesses amounts to giving a sharp estimate on the number of common zeros of the set of polynomials. Several such estimates are available. In particular, a result of Lang and Weil[1] (Theorem 2.2) that we will later use in the paper gives general lower and upper bounds on the number of rational points on varieties. Moreover, there are algorithmic results (see [1, 2, 7, 12, 14] and the references therein) that consider the problem of counting rational points on a given variety that belongs to a certain restricted class of varieties over finite fields.

Thus, it appears that the result of [8] already covers the variety sampling problem. However, this is not the case because of the following subtleties:

(1) Our relation is not necessarily self-reducible in the strong sense required by the construction of [8]. What required by this result is that partial fixings of the witness can be done in steps of at most logarithmic length (to allow for an efficient enumeration of all possible fixings). Namely, in our case, a partial fixing of x amounts to choosing

[1]This result can be seen as a consequence of the Weil theorem (initially conjectured in [18]) which is an analog of the Riemann hypothesis for curves over finite fields.

a particular value for one of the n variables. The portion of x corresponding to the variable being fixed would have length $\log q$, and in general, this can be much larger than $O(\log |x|)$.

(2) The general Lang-Weil estimate gives interesting bounds only when the underlying field is fairly large.

(3) The algorithmic results mentioned above, being mostly motivated by cryptographic or number-theoretic applications such as primality testing, focus on very restricted classes of varieties, for instance, elliptic [14] or hyperelliptic [1] curves (or general plane curves [7] that are only defined over a constant number of variables), or low-dimensional Abelian varieties [2]. Moreover, they are efficient in terms of the running time with respect to the logarithm of the field size and the dependence on the number of variables or the degree (whenever they are not restricted to constants) can be exponential.

Hence, over large fields, fine granularity of the self-reduction cannot be fulfilled and over small fields, no reliable and efficient implementation of a counting oracle is available for our problem, and we cannot directly apply the general sampler of [8]. In this work, we construct an efficient sampler that directly utilizes the algebraic structure of the problem. The main theorem that we prove is the following:

Theorem 1.1. (Main theorem) *Let the integer $k > 0$ be any absolute constant, $n > k$ and $d > 0$ be positive integers, $\epsilon > 0$ be an arbitrarily small parameter, and q be a large enough prime power. Suppose that $f_1, \ldots, f_k \in \mathbb{F}_q[x_1, \ldots, x_n]$ are polynomials, each of total degree at most d, whose set of common zeros defines an affine variety $V \subseteq \mathbb{F}_q^n$ of co-dimension k. There is a randomized algorithm that, given the description of $f_1, \ldots, f_k$ and the parameter ϵ, outputs a random point $v \in \mathbb{F}_q^n$ such that the distribution of v is $(6/q^{1-\epsilon})$-close to the uniform distribution on V. The worst case running time of the algorithm is polynomial in $n, d, \log q$, and the description length[2] of $f_1, \ldots, f_k$.*

Though we present the above result for affine varieties, our techniques can be readily applied to the same problem for projective varieties as well. At a high level, the algorithm is simple and intuitive, and can be roughly described as follows: To sample a point on a variety V of co-dimension k, we first sample a k-dimensional affine subspace A uniformly at random and then a random point on $V \cap A$. To make the analysis clear, we show (in Section 3) that the problem can be viewed as a sampling problem on *almost regular* bipartite graphs, where one can sample a left vertex almost uniformly by picking the left neighbor of a random edge. The main part of the analysis (Section 4) is to show why this reduction holds, and requires basic tools from Algebraic Geometry, in particular the Lang-Weil estimate on the number of points on varieties (Theorem 2.2), and details on how to deal with problems such as varying dimension and size of the intersection $V \cap A$. The reduction combined with the graph sampling algorithm constitutes the sampling algorithm claimed in the main theorem.

Connection with Randomness Extractors

Trevisan and Vadhan [17] introduced the notion of *samplable sources* as probability distributions that can be sampled using small, e.g., polynomial-sized, boolean circuits. An *extractor* for samplable sources is a deterministic function whose output, when the input is

[2]We consider an explicit description of polynomials given by a list of their nonzero monomials.

randomly chosen according to any samplable distribution, has a distribution that is statistically close to uniform. Assuming the existence of certain hard functions, they constructed such extractors.

As a natural class of samplable distributions, Dvir, Gabizon and Wigderson [6] considered the class of distributions that are samplable by low-degree multivariate polynomials. They gave a construction of extractors for such sources over sufficiently large finite fields that does not rely on any hardness assumption and achieves much better parameters. Moreover, they introduced the dual notion of *algebraic sources* that are defined as distributions that are uniform on rational points of low-degree affine varieties, and asked whether efficient extractors exist for such sources. Our main theorem shows that algebraic sources (for a wide range of parameters) are close to samplable distributions, and hence, any extractor for samplable distributions is also an extractor for such algebraic sources. Very recently, Dvir [5] gave a direct and unconditional construction of an extractor for algebraic sources when the field size is sufficiently large.

2. Preliminaries and Basic Facts

We will use a simple form of the well known Schwartz-Zippel lemma and a theorem by Lang and Weil for bounding the number of the points on a variety:

Lemma 2.1. (Schwartz-Zippel) [15, 19] *Let f be a nonzero n-variate polynomial of degree d defined over a finite field $\mathbb{F}_q$. Then the number of zeros of f is at most dq^{n-1}.* ∎

Theorem 2.2. (Lang-Weil) [10] *Let n, d, r be positive integers. There exists a constant $A(n,d,r)$ depending only on n, d, r such that for any irreducible r-dimensional variety V of degree d defined in a projective space $\mathbb{P}^n$ over a finite field $\mathbb{F}_q$, we have $|N - q^r| \leq (d-1)(d-2)q^{r-\frac{1}{2}} + A(n,d,r)q^{r-1}$, where N is the number of rational points of V over $\mathbb{F}_q$.* ∎

This theorem can be generalized to the case of reducible varieties as follows:

Corollary 2.3. *Let n, d, r be positive integers. There exists a constant $A'(n,d,r)$ depending only on n, d, r and a constant $\delta(d)$ depending only on d and integer s, $1 \leq s \leq d$, such that for any r-dimensional variety V of degree d defined in a projective space $\mathbb{P}^n$ over a finite field $\mathbb{F}_q$ we have $|N - sq^r| \leq \delta(d)q^{r-\frac{1}{2}} + A'(n,d,r)q^{r-1}$, where N is the number of rational points of V over $\mathbb{F}_q$.*

Proof. Let $V_1 \cup V_2 \cup \ldots \cup V_t$, where $1 \leq t \leq d$, be a decomposition of V into distinct irreducible components and denote the set of r-dimensional components in this decomposition by S. Let $s := |S|$. Note that each component $V_i \notin S$ has dimension at most $r-1$ and by Theorem 2.2, the number of points on the union of the components outside S is negligible, namely, at most $A''q^{r-\frac{3}{2}}$ where A'' is a parameter depending only on n, d, r. Hence to prove the corollary, it suffices to bound the number of points on the union of the components in S.

For each component $V_i \in S$ we can apply Theorem 2.2, which implies that the number of points of V_i in $\mathbb{P}^n$, assuming that its degree is d_i, is bounded from q^r by at most $(d_i - 1)(d_i - 2)q^{r-\frac{1}{2}} + \alpha_i q^{r-1}$, for some α_i that depends only on n, d_i, r. This upper bounds the number of points of V by

$$\sum_{i=1}^{s} |V_i| \leq sq^r + \delta_1 q^{r-\frac{1}{2}} + A_1 q^{r-1},$$

where $\delta_1 \stackrel{\text{def}}{=} \sum_{i=1}^{s}(d_i-1)(d_i-2) \le d^2$ (from the fact that $\sum_{i=1}^{s} d_i \le d$) and $A_1 \stackrel{\text{def}}{=} \sum_{i=1}^{s}\alpha_i$. Note that A_1 and δ_1 can be upper bounded by quantities depending only on n, d, r and d, respectively. This proves one side of the inequality.

For the lower bound on $|V|$, we note that the summation above counts the points at the intersection of two irreducible components multiple times, and it will be sufficient to discard all such points and lower bound the number of points that lie on exactly one of the components. Take a distinct pair of the irreducible components, V_i and V_j. The intersection of these varieties defines an $(r-1)$-dimensional variety, which by the upper bound we just obtained can have at most $s_{ij}q^{r-1} + \delta_2 q^{r-1.5} + A_2 q^{r-2}$) points, for some $s_{ij} \le d^2$, and parameters δ_2 depending only on d and A_2 depending on n, k, r. Hence, considering all the pairs, the number of points that lie on more than one of the irreducible components is no more than $\binom{d}{2}(d^2q^{r-1} + \delta_2 q^{r-1.5} + A_2 q^{r-2})$, which means that the number of distinct points of V is at least $\sum_{i=1}^{s}|V_i| - d^4q^{r-1} - d^2\delta_2 q^{r-\frac{3}{2}} - d^2A_2q^{r-2}$, which is itself at least $sq^r - \delta_1 q^{r-\frac{1}{2}} - (A_1+d^4)q^{r-1} - d^2\delta_2 q^{r-\frac{3}{2}} - d^2A_2q^{r-2}$. Taking (crudely) $A'(n,d,r) \stackrel{\text{def}}{=} A_1 + d^2A_2 + d^4 + d^2\delta_2 + A''$ and $\delta \stackrel{\text{def}}{=} \delta_1$ proves the corollary. ∎

Remark 2.4. Corollary 2.3 also holds for affine varieties. An affine variety V can be seen as the restriction of a projective variety $\bar{V}$ to the affine space, where no irreducible component of $\bar{V}$ is fully contained in the hyperplane at infinity. Then the *affine dimension* of V will be the (top) dimension of $\bar{V}$, and the bound in Corollary 2.3 holds for V if the affine dimension of the variety is taken as the parameter r in the bound. This is because each irreducible component of $\bar{V}$ intersects the hyperplane at infinity at a variety of dimension less than r, and by Theorem 2.2, adding those points to the estimate will have a negligible effect of order $q^{r-\frac{3}{2}}$.

Finally, we review some basic notions that we use from probability theory. The *statistical distance* (or *total variation distance*) of two distributions $\mathcal{X}$ and $\mathcal{Y}$ defined on the same finite space S is defined as $\frac{1}{2}\sum_{s\in S}|\Pr_{\mathcal{X}}(s) - \Pr_{\mathcal{Y}}(s)|$, where $\Pr_{\mathcal{X}}$ and $\Pr_{\mathcal{Y}}$ denote the probability measures on S defined by the distributions $\mathcal{X}$ and $\mathcal{Y}$, respectively. Note that this is half the ℓ_1 distance of the two distributions when regarded as vectors of probabilities over S. It can be shown that the statistical distance of the two distributions is at most ϵ if and only if for every $T \subseteq S$, we have $|\Pr_{\mathcal{X}}[T] - \Pr_{\mathcal{Y}}[T]| \le \epsilon$. When the statistical distance of $\mathcal{X}$ and $\mathcal{Y}$ is at most ϵ, we say that $\mathcal{X}$ and $\mathcal{Y}$ are ϵ-close. We will also use the notion of a convex combination of distributions, defined as follows:

Definition 2.5. Let $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_k$ be probability distributions on a finite set S and $\alpha_1, \alpha_2, \ldots, \alpha_k$ be nonnegative real values that sum up to 1. Then the *convex combination* $\alpha_1\mathcal{X}_1 + \alpha_2\mathcal{X}_2 + \cdots + \alpha_n\mathcal{X}_n$ is a distribution $\mathcal{X}$ on S given by the probability measure $\Pr_{\mathcal{X}}(x) \stackrel{\text{def}}{=} \sum_{i=1}^{k}\alpha_i \Pr_{\mathcal{X}_i}(x)$, for $x \in S$.

There is a simple connection between convex combinations and distance of distributions:

Proposition 2.6. *Let $\mathcal{X}, \mathcal{Y}$, and $\mathcal{E}$ be probability distributions on a finite set S such that for some $0 \le \epsilon \le 1$, $\mathcal{X} = (1-\epsilon)\mathcal{Y} + \epsilon\mathcal{E}$. Then $\mathcal{X}$ is ϵ-close to $\mathcal{Y}$.* ∎

3. A Vertex Sampling Problem

In this section we introduce a sampling problem on graphs, and develop an algorithm to solve it. We will later use this algorithm as a basic component in our construction of samplers for varieties. The problem is as follows:

Problem 3.1. Let G be a bipartite graph defined on a set $\mathcal{L}$ of left vertices and $\mathcal{R}$ of right vertices. Suppose that the degree of every vertex on the right is between 1 and d, for some $d > 1$, and the degree of every vertex on the left differs from an integer ℓ by at most $\delta\ell$. We are given an oracle $\mathsf{RSamp}(G)$ that returns an element of $\mathcal{R}$ chosen uniformly at random (and independently at each call), and an oracle $\mathsf{RNei}(v)$ that returns the neighbor list of a given vertex $v \in \mathcal{R}$. Construct an algorithm that outputs a random vertex in $\mathcal{L}$ almost uniformly.

Intuitively, for a bipartite graph which is regular from left and right, sampling a vertex on the left amounts to picking a random edge in the graph, which is in turn possible by choosing a random edge connected to a random vertex on the right side. Here of course, the graph is not regular, however the concentration of the left degrees around ℓ allows us to treat the graph as if it were regular and get an almost uniform distribution on $\mathcal{L}$ by picking a random edge. We will compensate the irregularity from right by using a "trial and error" strategy. The pseudocode given in Algorithm 1 implements this idea. The algorithm in fact handles a more general situation, in which a call to RSamp can fail (and return a special failure symbol $\perp$) with some probability upper bounded by a given parameter p.

Algorithm 1 BipartiteSample

Require: $G, \mathsf{RSamp}, \mathsf{RNei}$ given as in Problem 3.1, and p denoting the failure probability of RSamp.

1: Let δ, d be as in Problem 3.1.
2: $t_0 \leftarrow \lceil \frac{d}{1-p} \ln(\frac{1-\delta}{\delta}) \rceil$; $t \leftarrow t_0$
3: **while** $t \geq 0$ **do**
4: $\quad t \leftarrow t - 1$; $R \leftarrow \mathsf{RSamp}(G)$
5: $\quad$ **if** $R \neq \perp$ **then**
6: $\quad\quad V \leftarrow \mathsf{RNei}(R)$
7: $\quad\quad$ With probability $|V|/d$, output an element of V uniformly at random and return.
8: $\quad$ **end if**
9: **end while**
10: Output an arbitrary element of $\mathcal{L}$.

Lemma 3.2. *The output distribution of Algorithm 1 is supported on $\mathcal{L}$ and is $3\delta/(1-\delta)$-close to the uniform distribution on $\mathcal{L}$.*

Proof. First we focus on one iteration of the **while** loop in which the call to RSamp has not failed, and analyze the output distribution of the algorithm conditioned on the event that Line 7 returns a left vertex. In this case, one can see the algorithm as follows: Add a special vertex v_0 to the set of left vertices $\mathcal{L}$. Bring the degree of each right vertex up to d by connecting it to v_0 as many times as necessary. Hence, the graph G now becomes d-regular from right. Now the algorithm picks a random element $R \in \mathcal{R}$ and a random neighbor of R and independently repeats the process if v_0 is picked as a neighbor.

Let $T \subseteq \mathcal{L}$ be a non-empty subset of the left vertices (excluding v_0) in the graph. We want to estimate the probability of the event T. We can write this probability as follows:

$$\Pr[T] = \sum_{r\in\mathcal{R}} \Pr[T \mid R = r] \Pr[R = r] = \frac{1}{|\mathcal{R}|} \sum_{r\in\mathcal{R}} \Pr[T \mid R = r] = \frac{1}{d|\mathcal{R}|} \sum_{r\in\mathcal{R}} |T \cap \Gamma(r)|,$$

where in the last equation $\Gamma(r)$ is the set of neighbors of r in the graph. Hence the summation can be simplified as the number of edges connected to T. This quantity is in the range $|T|\ell(1 \pm \delta)$, because the left degrees are all concentrated around ℓ, ignoring v_0 which is by assumption not in T. That is,

$$\Pr[T] = \Pr[T, \neg v_0] = \frac{|T|\ell}{d|\mathcal{R}|}(1 \pm \delta), \tag{3.1}$$

where we use the shorthand $(1 \pm \delta)$ to denote a quantity in the range $[1 - \delta, 1 + \delta]$.

Hence the probabilities of all events that exclude v_0 are close to one another, which implies that the distribution of the outcome of a single iteration of the algorithm, conditioned on a non-failure, is close to uniform. We will now make this statement more rigorous.

The degree of v_0 can be estimated as

$$\deg(v_0) = d|\mathcal{R}| - |\mathcal{L}|\ell(1 \pm \delta)$$

by equating the number of edges on the left and right side of the graph. Similar to what we did for computing the probability of T we can compute the probability of picking v_0 as

$$\Pr(v_0) = \frac{1}{d|\mathcal{R}|} \deg(v_0) = 1 - \frac{|\mathcal{L}|}{d|\mathcal{R}|}\ell(1 \pm \delta).$$

Combining this with (3.1) we get that

$$\Pr[T \mid \neg v_0] = \frac{\Pr[T, \neg v_0]}{1 - \Pr(v_0)} = \frac{|T|}{|\mathcal{L}|}\left(1 \pm \frac{2\delta}{1 - \delta}\right).$$

Hence, the output distribution of a single iteration of the **while** loop, conditioned on a non-failure (i.e., the event that the iteration reaches Line 7 and outputs an element of $\mathcal{L}$) is $2\delta/(1 - \delta)$-close to the uniform distribution on $\mathcal{L}$. Now denote by φ the failure probability. To obtain an upper bound on φ, note that the probability of sampling v_0 at Line 7 of the algorithm is at most $(d - 1)/d$ since each vertex on the right has at least one neighbor different from v_0. Hence,

$$\varphi \le 1 - (1 - p)/d \tag{3.2}$$

Now we get back to the whole algorithm, and notice that if the **while** loop iterates for up to t_0 times, the output distribution of the algorithm can be written as a convex combination

$$\mathcal{O} = (1 - \varphi)\mathcal{D} + (1 - \varphi)\varphi\mathcal{D} + \cdots + (1 - \varphi)\varphi^{t_0-1}\mathcal{D} + \varphi^{t_0}\mathcal{E} = (1 - \varphi^{t_0})\mathcal{D} + \varphi^{t_0}\mathcal{E},$$

where $\mathcal{D}$ is the output distribution of a single iteration conditioned on a non-failure and $\mathcal{E}$ is an arbitrary *error distribution* corresponding to the event that the algorithm reaches the last line. The coefficient of $\mathcal{E}$, for $t_0 \ge \frac{d}{1-p}\ln(\frac{1-\delta}{\delta})$, can be upper bounded using (3.2) by

$$\varphi^{t_0} \le \left(1 - \frac{1 - p}{d}\right)^{\frac{d}{1-p}\ln(\frac{1-\delta}{\delta})} \le \frac{\delta}{1 - \delta}.$$

This combined with the fact that $\mathcal{D}$ is $2\delta/(1 - \delta)$-close to uniform and Proposition 2.6 implies that $\mathcal{O}$ is $3\delta/(1 - \delta)$-close to the uniform distribution on $\mathcal{L}$. ∎

4. Sampling Rational Points on Varieties

Now we are ready to describe and analyze our algorithm for sampling rational points on varieties. For the sake of brevity, we will present the results in this section for affine varieties. However, they can also be shown to hold for projective varieties using similar arguments.

We reduce the problem to the vertex sampling problem described in the preceding section. The basic idea is to intersect the variety with randomly chosen affine spaces in $\mathbb{F}_q^n$ and narrowing-down the problem to the points within the intersection. Accordingly, the graph G in the bipartite sampling problem will be defined as the incidence graph of the points on the variety with affine spaces. This is captured in the following definition:

Definition 4.1. Let V be an affine variety of co-dimension k in $\mathbb{F}_q^n$. Then the *affine incidence graph* of the variety is a bipartite graph $G = (L \cup R, E)$ defined as follows:

- The left vertex set is V,
- For a k-dimensional affine space A, we say that A *properly intersects* V if the intersection $V \cap A$ is non-empty and has dimension zero. Then the right vertex set of G is defined as the set of k-dimensional affine spaces in $\mathbb{F}_q^n$ that properly intersect V.
- There is an edge between $u \in L$ and $v \in R$ if and only if the affine space v contains the point u.

Before utilizing the vertex sampling algorithm of the preceding section, we need to develop the tools needed for showing that the affine incidence graph satisfies the properties needed by the algorithm. We begin with an estimate on the number of linear and affine subspaces of a given dimension. The estimate is straightforward to obtain, yet we include a proof for completeness.

Proposition 4.2. *Let $\mathbb{F}$ be a finite field of size $q \geq \sqrt{2k}$, and let N_1 and N_2 be the number of distinct k-dimensional linear and affine subspaces of $\mathbb{F}^n$, respectively. Then we have*

(1) $|N_1/q^{k(n-k)} - 1| \leq 2k/q^2$,
(2) $|N_2/q^{(k+1)(n-k)} - 1| \leq 2k/q^2$.

Proof. If $k = n$, then $N_1 = N_2 = 1$, and the claim is obvious. Hence, assume that $k < n$. Denote by $N_{k,n}$ the number of ways to choose k linearly independent vectors in $\mathbb{F}^n$. That is, $N_{k,n} = (q^n - 1)(q^n - q) \cdots (q^n - q^{k-1})$. This quantity is upper bounded by q^{nk}, and lower bounded by $(q^n - q^{k-1})^k \geq q^{nk}(1 - kq^{k-1-n}) \geq q^{nk}(1 - k/q^2)$. Hence, the reciprocal of $N_{k,n}$ can be upper bounded as follows:

$$\frac{1}{N_{k,n}} \leq \frac{q^{-nk}}{1 - k/q^2} = q^{-nk}\left(1 + \frac{k}{q^2} \cdot \frac{1}{1 - k/q^2}\right) \leq q^{-nk}\left(1 + \frac{2k}{q^2}\right),$$

where the last inequality follows from the assumption that $q^2 \geq 2k$.

The number of k-dimensional subspaces of $\mathbb{F}^n$ is the number of ways one can choose k linearly independent vectors in $\mathbb{F}^n$, divided by the number of bases a k-dimensional vector space can assume. That is, $N_1 = N_{k,n}/N_{k,k}$. By the bounds above, we obtain

$$N_1 \leq q^{nk} \cdot q^{-k^2}(1 + 2k/q^2) \qquad \text{and} \qquad N_1 \geq q^{nk}(1 - k/q^2) \cdot q^{-k^2},$$

which implies $|N_1/q^{k(n-k)} - 1| \leq 2k/q^2$. The second part of the claim follows from the observation that two translations of a k-dimensional subspace A defined by vectors u and

v coincide if and only if $u - v \in A$. Hence, the number of affine k-dimensional subspaces of $\mathbb{F}^n$ is the number of k-dimensional subspaces of $\mathbb{F}^n$ multiplied by the number of cosets of A, i.e., $N_2 = N_1 q^{n-k}$. ∎

The following two propositions show that a good fraction of all k-dimensional affine spaces properly intersect any affine variety of co-dimension k.

Proposition 4.3. *Let n, d, k be positive integers, and $V \subset \mathbb{F}_q^n$ be an affine variety of co-dimension k defined by the zero-set of k polynomials $f_1, \ldots, f_k \in \mathbb{F}_q[x_1, \ldots, x_n]$, each of degree at most d. Suppose that $v \in V$ is a fixed point of V. Then the fraction of k-dimensional affine spaces passing through v that properly intersect V is at least $1 - B(k, n, d)/q$, where $B(n, d, k)$ is independent of q and polynomially large in n, d, k.*

Proof. Without loss of generality, assume that v is the origin, and that $q \geq \sqrt{2k}$. Denote by L the set of k-dimensional linear subspaces that can be parametrized as

$$\begin{pmatrix} x_{k+1} \\ x_{k+2} \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \alpha_{11} & \ldots & \alpha_{1k} \\ \alpha_{21} & \ldots & \alpha_{2k} \\ \vdots & \ddots & \vdots \\ \alpha_{(n-k)1} & \ldots & \alpha_{(n-k)k} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix},$$

where $\alpha \stackrel{\text{def}}{=} \{\alpha_{11}, \ldots, \alpha_{(n-k)k}\}$ is a set of indeterminates in $\mathbb{F}_q$. Note that $|L| = q^{|\alpha|} = q^{k(n-k)}$, and define the polynomial ring $\mathcal{R} \stackrel{\text{def}}{=} \mathbb{F}_q[\alpha_{11}, \ldots, \alpha_{(n-k)k}]$. We first upper bound the number of *bad* subspaces in L whose intersections with V have nonzero dimensions. Substituting the linear forms defining $x_{k+1}, \ldots, x_n$ in $f_1, \ldots, f_k$ we see that the intersection of V and the elements of L is defined by the common zero-set of polynomials $g_1, \ldots, g_k \in \mathcal{R}[x_1, \ldots, x_k]$, where for each $i \in [k]$,

$$g_i(x_1, \ldots, x_k) \stackrel{\text{def}}{=} f_i(x_1, \ldots, x_k, \alpha_{11}x_1 + \cdots + \alpha_{1k}x_k, \ldots, \alpha_{(n-k)1}x_1 + \cdots + \alpha_{(n-k)k}x_k).$$

Each g_i, as a polynomial in $x_1, \ldots, x_k$, has total degree at most d and each of its coefficients is a polynomial in $\alpha_{11}, \ldots, \alpha_{(n-k)k}$ of total degree at most d. Denote by $I \subseteq \mathcal{R}[x_1, \ldots, x_k]$ the ideal generated by $g_1, \ldots g_k$. For every $j \in [n]$, the ideal $I \cap \mathcal{R}[x_j]$ is generated by a polynomial h_j. Each coefficient of h_j can be written as a polynomial in $\mathcal{R}$ with total degree at most D, where for a fixed k, D is polynomially large in d. This can be shown using an elimination method, e.g., generalized resultants or Gröbner bases (cf. [4, 11, 9]). Take any coefficient of h_j which is a nonzero polynomial in $\mathcal{R}$. The number of the choices of α which makes this coefficient zero is, by Lemma 2.1, at most $Dq^{k(n-k)-1}$. This also upper bounds the number of the choices of α that make h_j identically zero.

A union bound shows that for all but at most $nDq^{k(n-k)-1}$ choices of α none of the polynomial h_j is identically zero, and hence the solution space of $g_1, \ldots, g_k$ is zero dimensional (and obviously non-empty, as we already know that it contains v). This gives an upper bound of nD/q on the fraction of bad subspaces in L.

By Proposition 4.2, the set L contains at least a $1 - 2k/q^2$ fraction of all k-dimensional subspaces of $\mathbb{F}_q^n$. Hence, the fraction of k-dimensional subspaces of $\mathbb{F}_q^n$ that properly intersect V is at least

$$\left(1 - \frac{2k}{q^2}\right)\left(1 - \frac{nD}{q}\right) \geq \left(1 - \frac{2k + nD}{q}\right).$$

The claim follows by taking $B \stackrel{\text{def}}{=} 2k + nD$. ∎

Proposition 4.4. *Let k, n, d be positive integers, and $V \subset \mathbb{F}_q^n$ be an affine variety of co-dimension k defined by the zero-set of k polynomials $f_1, \ldots, f_k \in \mathbb{F}_q[x_1, \ldots, x_n]$, each of degree at most d. The fraction of k-dimensional affine subspaces that properly intersect V is at least*

$$d^{-k}\left(1 - \frac{\delta(d)}{\sqrt{q}} - \frac{A'(n,d,n-k) + B(n,d,k)}{q}\right),$$

where $\delta(\cdot), A'(\cdot), B(\cdot)$ are as in Corollary 2.3 and Proposition 4.3.

Proof. We use a counting argument to obtain the desired bound. Denote by N, N_1, and N_2 the number of points of V, the number of k-dimensional subspaces and k-dimensional affine subspaces in $\mathbb{F}_q^n$, respectively. Then Corollary 2.3 (followed by Remark 2.4) implies that

$$N \geq sq^{n-k} - \delta(d)q^{n-k-\frac{1}{2}} - A'(n,d,n-k)q^{n-k-1},$$

for some $s \in [d^k]$ (as the degree of V is at most d^k).

By Proposition 4.3, for every $v \in V$, at least $N_1(1 - B(n,d,k)/q)$ affine subspace pass through v and properly intersect V. Hence in total $N \cdot N_1(1 - B(n,d,k)/q)$ affine spaces properly intersect V, where we have counted every such affine space at most d^k times (This is because the intersection of V and an affine space A that properly intersects it is of size at most d^k, and A is counted once for each point at the intersection). Thus, the fraction of distinct affine subspaces that properly intersect V is at least

$$\frac{NN_1(1 - B(n,d,k)/q)}{d^k N_2}.$$

By the fact that $N_2 = N_1 q^{n-k}$ and the lower bound on N, we conclude that this fraction is at least

$$d^{-k}\left(s - \frac{\delta(d)}{\sqrt{q}} - \frac{A'(n,d,n-k)}{q}\right)\left(1 - \frac{B(n,d,k)}{q}\right).$$

As $s \geq 1$, this proves the claim. ∎

Now having the above tools available, we are ready to give the reduction from variety sampling to the vertex sampling problem introduced in the preceding section and prove our main theorem:

Proof of Theorem 1.1. Let $G = (L \cup R, E)$ be the affine incidence graph of V. We will use Algorithm 1 on G. To show that the algorithm works, first we need to implement the oracles RSamp and RNei that are needed by the algorithm.

The function RSamp simply samples a k-dimensional affine space of $\mathbb{F}_q^n$ uniformly at random, and checks whether the outcome A properly intersects V. To do so, one can parametrize the affine subspace as in the proof of Proposition 4.3 and substitute the parametrization in $f_1, \ldots, f_k$ to obtain a system of k polynomial equations in k unknowns, each of degree at most D which is polynomially large in d. As k is an absolute constant, it is possible to solve this system in polynomial time using multipolynomial resultants or the Gröbner bases method combined with backward substitutions. If at any point, the elimination of all but any of the variables gives the zero polynomial, it turns out that the system does not define a zero-dimensional variety and hence, A does not properly intersect V. Also, if the elimination results in a univariate polynomial that does not have a solution in $\mathbb{F}_q$, the intersection becomes empty, again implying that A does not properly intersect V. In both cases RSamp fails, and otherwise, it outputs A. Furthermore, if the intersection is proper,

the elimination method gives the list of up to D^k points at the intersection, which one can use to construct the oracle RNei.

Now we need to show that the graph G satisfies the conditions required by Lemma 3.2. By the argument above, the degree of every right vertex in G is at least 1 and at most D^k, which is polynomially large in d. Let p denote the failure probability of RSamp. Then Proposition 4.4 implies that $p \le d^{-k}/2$ when $q \ge \max\{16\delta^2(d), 4(A'(n,d,n-k) + B(n,d,k))\}$.

To bound the left degrees of the graph, note that each left node, which is a point on V, is connected to all k-dimensional affine subspaces that properly intersect V and pass through the point. The number of such spaces is, by Proposition 4.2, at most $q^{k(n-k)}(1 + 2k/q^2)$ (assuming $q \ge \sqrt{2k}$), and by combination of Proposition 4.2 and Proposition 4.3, at least

$$q^{k(n-k)} \left(1 - \frac{2k}{q^2}\right) \left(1 - \frac{B(k,n,d)}{q}\right) \ge q^{k(n-k)} \left(1 - \frac{2k + B(k,n,d)}{q}\right).$$

Now if we choose $q \ge (2k + B(k,n,d))^{1/\epsilon}$, the left degrees become concentrated in the range $q^{k(n-k)}(1 \pm 1/q^{1-\epsilon})$.

Putting everything together, now we can apply Lemma 3.2 to conclude that the output distribution of the algorithm is $(6/q^{1-\epsilon})$-close to the uniform distribution on V.

To show the efficiency of the algorithm, first note that Algorithm 1 calls each of the oracles RSamp and RNei at most

$$\frac{D^k}{1-p} \ln \left(\frac{1 - q^{\epsilon-1}}{q^{\epsilon-1}}\right) \le 2D^k(1-\epsilon)\ln q$$

times, which is upper bounded by a polynomial in $d, \ln q$. Hence it remains to show that the implementation of the two oracles are efficient. The main computational cost of these functions is related to the problem of deciding whether a system of k polynomial equations of bounded degree in k unknowns has a zero dimensional solution space, and if so, computing the list of at most D^k solutions of the system. As in our case k is a fixed constant, elimination methods can be efficiently applied to reduce the problem to that of finding the zero-set of a single uni-variate polynomial of bounded degree. A randomized algorithm is given in [13] for this problem that runs in expected polynomial time. Thus, we can use this algorithm as a sub-routine in RSamp and RNei to get a sampling algorithm that runs in expected polynomial time. Then it is possible to get a worst-case polynomial time algorithm by using a *time-out* trick, i.e., if the running time of the sampler exceeds a (polynomially large) threshold, it is forced to terminate and output an arbitrary point in $\mathbb{F}_q^n$. The error caused by this can increase the distance between the output distribution of the sampler and the uniform distribution on V by a negligible amount that can be made arbitrarily small (and in particular, smaller than $1/q^{1-\epsilon}$), and hence, is of little importance.

Finally, we need an efficient implementation of the field operations over $\mathbb{F}_q$. This is again possible using the algorithm given in [13]. Moreover, when the characteristic of the field is small, deterministic polynomial time algorithms are known for this problem [16]. ∎

5. Concluding Remarks

We showed the correctness and the efficiency of our sampling algorithm for varieties of constant co-dimension over large fields. Though our result covers important special cases such as sampling random roots of multivariate polynomials, relaxing either of these requirements is an interesting problem. In particular, it remains an interesting problem

to design samplers that work for super-constant (and even more ambitiously, linear in n) co-dimensions (in our result, the dependence of the running time on the co-dimension is exponential and thus, we require constant co-dimensions). Moreover, in this work we did not attempt to optimize or obtain concrete bounds on the required field size, which is another interesting problem. Finally, the error of our sampler (i.e., the distance of the output distribution from the uniform distribution on the variety) depends on the field size, and it would be interesting to bring down the error to an arbitrary parameter that is given to the algorithm.

Acknowledgment

We would like to thank Frédéric Didier for fruitful discussions.

References

[1] L. Adleman and M-D. Huang. Primality testing and abelian varieties over finite fields. *Springer Lecture Notes in Mathematics*, 1512, 1992.

[2] L. Adleman and M-D. Huang. Counting rational points on curves and abelian varieties over finite fields. *Springer LNCS (Proc. Algorithmic Number Theory Symposium)*, 1122:1–16, 1996.

[3] M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of NP-witnesses using an NP-oracle. *Inform. and Comp.*, 163:510 –526, 2000.

[4] D. A. Cox, J. B. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer, 1992.

[5] Z. Dvir. Deterministic extractors for algebraic sources. Unpublished manuscript, 2008.

[6] Z. Dvir, A. Gabizon, and A. Wigderson. Extractors and rank extractors for polynomial sources. In *Proc. 48th FOCS*, page 52–62, 2007.

[7] M-D. Huang and D. Ierardi. Counting rational points on curves over finite fields. *J. Symbolic Computation*, page 1–21, 1998.

[8] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43(2-3):169–188, 1986.

[9] D. Kapur and Y. N. Laksman. Elimination methods: An introduction, 1992. In *Symbolic and Numerical Computation for Artificial Intelligence* (B. Donald, D. Kapur and J. Mundy, editors), Academic Press.

[10] S. Lang and A. Weil. Number of points of varieties in finite fields. *American Journal of Mathematics*, 76(4):819–827, 1954.

[11] F. S. Macaulay. On some formulae in elimination. In *Proc. London Mathematical Society*, page 3–27, 1902.

[12] J. Pila. Frobenius maps of abelian varieties and finding roots of unity in finite fields. *Mathematics of Computation*, 55:745–763, 1990.

[13] M. Rabin. Probabilistic algorithms in finite fields. *SIAM Journal on Computing*, 9(2):273–280, 1980.

[14] R. Schoof. Elliptic curves over finite fields and the computation of square roots $\mod p$. *Mathematics of Computation*, page 483–494, 1985.

[15] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[16] V. Shoup. New algorithms for finding irreducible polynomials over finite fields. *Mathematics of Computation*, 54:435–447, 1990.

[17] L. Trevisan and S. Vadhan. Extracting randomness from samplable distributions. In *Proc. 41st FOCS*, page 32–42, 2000.

[18] A. Weil. Number of solutions of equations in finite fields. *Bulletin of the AMS*, 55(5):497–508, 1949.

[19] R. E. Zippel. Probabilistic algorithms for sparse polynomials. *Springer LNCS (EUROSCAM'79)*, 72:216–226, 1979.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 289–300
www.stacs-conf.org

REVERSE ENGINEERING PREFIX TABLES

JULIEN CLÉMENT [1] AND MAXIME CROCHEMORE [2,3] AND GIUSEPPINA RINDONE [3]

[1] GREYC, CNRS-UMR 6072, Université de Caen, 14032 Caen, France

[2] King's College London, Strand, London WC2R 2LS, UK

[3] Institut Gaspard-Monge, CNRS-UMR 8049, Université de Paris-Est
E-mail address: clement@info.unicaen.fr,maxime.crochemore@kcl.ac.uk,rindone@univ-mlv.fr

Abstract. The Prefix table of a string reports for each position the maximal length of its prefixes starting here. The Prefix table and its dual Suffix table are basic tools used in the design of the most efficient string-matching and pattern extraction algorithms. These tables can be computed in linear time independently of the alphabet size.

We give an algorithmic characterisation of a Prefix table (it can be adapted to a Suffix table). Namely, the algorithm tests if an integer table of size n is the Prefix table of some word and, if successful, it constructs the lexicographically smallest string having it as a Prefix table. We show that the alphabet of the string can be bounded to $\log_2 n$ letters. The overall algorithm runs in $O(n)$ time.

1. Introduction

The Prefix table of a string reports for each position on the string the maximal length of its prefixes starting at that position. The table stores the same information as the Border Table of the string, which memorises for each position the maximal length of prefixes ending at that position. Both tables are useful in several algorithms on strings. They are used to design efficient string-matching algorithms and are essential for this type of applications (see for example [15] or [6]).

The dual notion of the Prefix table, the Suffix Table (not to be confused with the Suffix Array that is related to the lexicographic order of the suffixes), simplifies the design of the pattern preprocessing for Boyer-Moore string-matching algorithm ([4], see also [6]). This preprocessing problem is notorious for its several unsuccessful attempts. Gusfield makes it a fundamental element of string-matching methods he presents in [15]. His Z-algorithm corresponds to the computation of the Suffix table. Along the same line, the Suffix table is an essential element of the Apostolico-Giancarlo string-matching algorithm [1], one of the ultimate improvements of the Boyer-Moore strategy: the maximal number of symbol comparisons is no more than $3n/2$ in the worst-case [9], which is half the number

Key words and phrases: design and analysis of algorithms, algorithms on strings, pattern matching, string matching, combinatorics on words, Prefix table, Suffix table.

This work is funded partially by CNRS and by the European Project AutoMathA.

of the original searching algorithm, and this is linked to the total running time of the algorithm. By the way, the technique is still the object of a conjecture about the polynomial number of configurations encountered during a search (see [17, 2]). These techniques or some reductions of them are the best choice for implementing search tools inside text editors or analogue software.

The computation of the table of borders is a classical question. It is inspired by an algorithm of Morris and Pratt (1970), which is at the origin of the first string-matching algorithm running in linear time independently of the alphabet size [17]. Despite the fact that Border Tables and Prefix tables contains the same information on the underlying string, the efficient algorithms to compute them are quite different (see [6, Chapter 1]). The first algorithm has a recursive expression while this is not so for the second one.

The technique used to compute the Prefix table works online on the string and takes advantage on what has been done on shorter prefixes of the string. It needs only simple combinatorial properties to be proved. A similar technique is used by Manacher [18] for finding the shortest even palindrome prefix of a string, which extends to a linear-time decomposition of a string into even palindromes ([14], see also [17]).

In this article we consider the reverse engineering problem: testing if a table of integers is the Prefix table of some string, and if so, producing such a string.

The same question can be stated for other data structures storing information on strings. Successful solutions then provide if-and-only-if conditions on the data structures and then complete characterisations, which is of main theoretical interest. It is also of interest for software testing problem related to the data structures when it comes to implementing them. It helps also design methods for randomly generating the data structures in relation to the previous problem.

As far as we know, reverse engineering has been first considered for Border Tables by Franek et al. [12]. Their algorithm tests whether an integer array is the Border Table of a string or not, and exhibits a corresponding string if it is. They solve the question in linear time, according to the size of the input array, for an unbounded alphabet. A refinement of the technique by Duval et al. [10, 11] solves the question for a bounded-size alphabet. Bannai et al. [3] characterise three other data structures intensively used in String Algorithmics: Directed Acyclic Subsequence Graph, Directed Acyclic Word Graph or Minimal Suffix Automaton, and Suffix Array. Their three testing algorithms run in linear time as well. Reconstructing a Suffix Array is obvious on a binary alphabet especially if a special symbol is appended to the string. For general alphabets, another solution is by Franek et al. [13]. In this situation, the algorithm accounts for descents in permutations (see [5]).

The case of Prefix table (or Suffix table) seems more difficult. However we get, as for previous questions, a linear-time test. The algorithm can provide the largest initial part of the array that is compatible with a Prefix table. When the array is a Prefix table it infers a string corresponding to it on the smallest possible alphabet. We show that indeed $\log_2 n$ letters are enough for a table of size n. In the algorithm we make use of a variable to store a set of letters that are forbidden in a certain context. It is surprising and remarkable that, due to combinatorial properties, the algorithm requires no sophisticated data structure to implement this variable and still runs in linear time.

Reverse engineering for the Longest Previous Factor table, which is an important component of Ziv-Lempel text compression method (see [7, 8]), is an open question. It is not known if a linear-time solution exists.

In next sections we first describe properties of Prefix tables, then design the testing algorithm, and finally analyse it.

2. Properties of Prefix tables

After basic definitions we state the major properties of Prefix tables.

2.1. Preliminaries and definitions

Let A be an ordered alphabet, $A = \{a_0, a_1, \ldots\}$. A word w of length $|w| = n$ is a finite sequence $w[0]w[1]\ldots w[n-1] = w[0\,..\,n-1]$ of letters of A. The language of all words is A^*, and A^+ is the set of nonempty words. For $0 \le i, j < n$ we say that the factor $w[i\,..\,j]$ occurs at position i on w. By convention, $w[i\,..\,j]$ is the empty word ε if $i > j$. The prefix (resp. suffix) of length ℓ, $0 \le \ell \le n$, of w is the word $u = w[0\,..\,\ell-1]$ (resp. $u = w[n-\ell\,..\,n-1]$). A border u of w is a word that is both a prefix and a suffix of w distinct from w itself.

Definition 2.1 (Prefix table)**.** The Prefix table Pref_w of a word $w \in A^+$ of length n, is the array of size n defined, for $0 \le i < n$, by

$$\mathrm{Pref}_w[i] = \mathrm{lcp}(w, w[i\,..\,n-1]),$$

where $\mathrm{lcp}(u, v)$ denotes the length of the longest common prefix of two words u and v.

Definition 2.2 (Extent and Anchor)**.** On a table t of size n we define, for $0 < i \le n$:

- the (right) *extent* $\mathrm{Extent}(t, i)$ of position i (according to t) is the integer
$$\mathrm{Extent}(t, i) = \max\{j + t[j] \mid 0 < j < i\} \cup \{i\},$$
- the *anchor* $\mathrm{Anchor}(t, i)$ of position i (according to t) is the set of positions on t reaching the extent of i, i.e.,
$$\mathrm{Anchor}(t, i) = \{f \mid 0 < f < i \text{ and } f + t[f] = \mathrm{Extent}(t, i)\}.$$

Remark 2.3. Let us consider two positions i and j on the Prefix table t, $i < j$, that satisfy $\mathrm{Extent}(t, i) \neq \mathrm{Extent}(t, j)$. Then, $\mathrm{Anchor}(t, i) \cap \mathrm{Anchor}(t, j) = \emptyset$. Indeed in this case $\mathrm{Extent}(t, i) < \mathrm{Extent}(t, j)$, and we have $\max \mathrm{Anchor}(t, i) < i$ while $\min \mathrm{Anchor}(t, j) \ge i$.

In the description of the algorithm and its analysis we refer to the set of letters following immediately the prefix occurrences of the borders of a word w. For $0 < i \le k \le n$, we note

$$\mathcal{E}(w, i, k) = \{w[|u|] \mid u \text{ is a border of } w[0\,..\,k-1] \text{ and } |u| > k - i\},$$

and we simply write $\mathcal{E}(w, k)$ when considering $\mathcal{E}(w, k, k)$.

Example. Let w be the word `ababaababababa`. The following table shows its Prefix table t, the values $\mathrm{Extent}(t, i)$, and the sets $\mathrm{Anchor}(t, i)$, for all values of i.

i	0	1	2	3	4	5	6	7	8	9	10	11	12
$w[i]$	a	b	a	b	a	a	b	a	b	a	b	a	–
$t[i]$	12	0	3	0	1	5	0	5	0	3	0	1	–
$\mathrm{Extent}(t, i)$	–	1	2	5	5	5	10	10	12	12	12	12	12
$\mathrm{Anchor}(t, i)$	–	$\emptyset$	$\emptyset$	$\{2\}$	$\{2\}$	$\{2,4\}$	$\{5\}$	$\{5\}$	$\{7\}$	$\{7\}$	$\{7,9\}$	$\{7,9\}$	$\{7,9,11\}$

The whole word has three nonempty borders: `ababa`, `aba`, and `a` occurring at respective positions 7, 9, and 11. We have $\mathcal{E}(w, 9, 12) = \{\mathtt{a}\}$ because $u = \mathtt{ababa}$ is the only border of w of length larger than $3 = 12 - 9$ and $w[|u|] = \mathtt{a}$. We have $\mathcal{E}(w, 12) = \{\mathtt{a}, \mathtt{b}\}$ because

the three nonempty borders of w are `ababa`, `aba`, and `a`, and because $w[5] = \{\texttt{a}\}$ and $w[3] = w[1] = \{\texttt{b}\}$.

Definition 2.4 (Validity and compatibility)**.** Let t be a table of size n. For $0 < i < n$, we say that:

- the table t is *valid until* i if there exists a word w for which $t[1\,..\,i] = \mathrm{Pref}_w[1\,..\,i]$. We also say that t is *compatible with* w *until* i.
- The table t is *valid* if there exists a word w such that $t = \mathrm{Pref}_w$ (note that in particular $t[0] = n$). We also say that t is *compatible with* w.

Example. Let t and t' be the two tables defined by:

i	0	1	2	3	4	5
$t[i]$	6	0	0	2	0	1
$t'[i]$	6	0	0	2	1	1

The table t is valid since it is compatible with `abcaba` for example. But the table t' is not valid since it cannot be the Prefix table of any word. However, t' is compatible with `abcaba` until position 3.

2.2. Properties

In this subsection we state if-and-only-if conditions on values appearing in a Prefix table. The algorithm of the next section is derived from them.

By definition, the values in the Prefix table of a word w of length n satisfy

$$0 \le \mathrm{Pref}_w[i] \le n - i \text{ and } \mathrm{Pref}_w[0] = n\ .$$

There is another simple property coming from the definitions of Extent and $\mathcal{E}$: when $g = \mathrm{Extent}(\mathrm{Pref}_w, i) = i$ and $i < n$, $w[i] \notin \mathcal{E}(w, g)$.

The next statement gives less obvious conditions (some of them are in [6]).

Proposition 2.5 (Necessary and sufficient conditions on values in a prefix table)**.** *Let w be a word of length n and Pref_w its Prefix table. Let $i \in \{1, \ldots, n-1\}$ and $g = \mathrm{Extent}(\mathrm{Pref}_w, i)$, and assume $i < g$. Then, for all $f \in \mathrm{Anchor}(\mathrm{Pref}_w, i)$, one has*

(i) $\mathrm{Pref}_w[i] < g - i$ *if and only if* $\mathrm{Pref}_w[i - f] < g - i$ *and then both* $\mathrm{Pref}_w[i] = \mathrm{Pref}_w[i - f]$ *and* $w[i - f + \mathrm{Pref}_w[i]] = w[i + \mathrm{Pref}_w[i]] \neq w[\mathrm{Pref}_w[i]]$.

(ii) $\mathrm{Pref}_w[i] = g - i$ *if and only if* $\begin{cases} \mathrm{Pref}_w[i - f] \ge g - i \text{ and} \\ g = n \text{ or } w[g] \neq w[g - i]\ . \end{cases}$

(iii) $\mathrm{Pref}_w[i] = g - i + \ell$ *and* $\ell > 0$ *if and only if*

$$\begin{cases} \mathrm{Pref}_w[i - f] = g - i \text{ and} \\ w[g\,..\,g + \ell - 1] = w[g - i\,..\,g - i + \ell - 1] \text{ and} \\ g + \ell = n \text{ or } w[g + \ell] \neq w[g - i + \ell]\ . \end{cases}$$

Proof. Let $f \in \mathrm{Anchor}(\mathrm{Pref}_w, i)$. Let $u = w[f\,..\,g-1]$ and $v = w[i\,..\,g-1]$. By definition of f and g, the word u is the longest prefix of w beginning at position f. This ensures that $w[g] \neq w[g - f]$ if $g < n$. The suffix v of u has another occurrence at position $i - f$ and one has $|v| = g - i$. With the notation, we get the following.

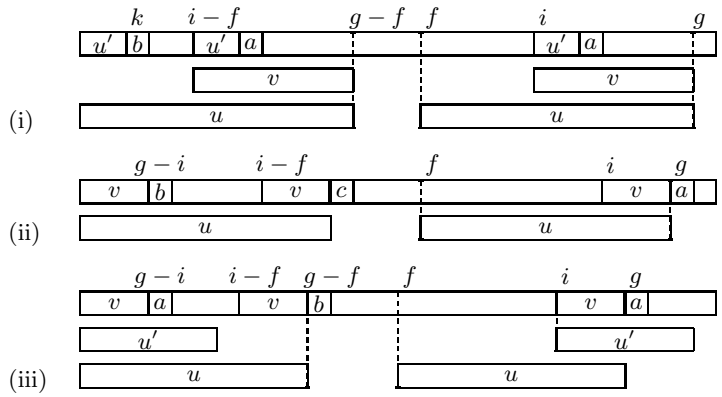


Figure 1: Illustration for the proof of Proposition 2.5. (**i**) Case $\mathrm{Pref}_w[i] < g - i$. In word w, the letter following u' at position $i - f$ is the same as the letter following it at i, and different from the letter at position $k = \mathrm{Pref}_w[i]$ ($a \neq b$). (**ii**) Case $\mathrm{Pref}_w[i] = g - i$. In word w, we have $\mathrm{Pref}_w[i - f] \geq g - i$. If $w[g]$ is defined and equals a, we have both $a \neq b$ and $a \neq c$. If in addition $b \neq c$, then $\mathrm{Pref}_w[i - f] = g - i$, otherwise, $\mathrm{Pref}_w[i - f] > g - i$. (**iii**) Case $\mathrm{Pref}_w[i] > g - i$. In word w, we have $a \neq b$ and then $\mathrm{Pref}_w[i - f] = g - i$.

(i) When $\mathrm{Pref}_w[i] = k < g - i = |v|$, one has $\mathrm{Pref}_w[i - f] = k$ (and the converse is also true). Moreover, $w[i\,..\,i + k - 1] = w[i - f\,..\,i - f + k - 1] = w[0\,..\,k - 1]$ and $w[i + k] = w[i - f + k] \neq w[k]$ (see Figure 1(i)).

(ii) Suppose $\mathrm{Pref}_w[i] = g - i$. Then $w[g] \neq w[g - i]$, if $g \neq n$, by definition of Pref_w. The word v being a suffix of u also occurs at position $i - f$, which implies $\mathrm{Pref}_w[i - f] \geq g - i$ as expected. Conversely, if $\mathrm{Pref}_w[i - f] \geq g - i$ then v is a prefix of w that occurs at position i. Since $w[g] \neq w[g - i]$ when $g < n$, one has $\mathrm{Pref}_w[i] = |v| = g - i$ (see Figure 1(ii)). The conclusion holds obviously if $g = n$.

(iii) Suppose $\mathrm{Pref}_w[i] = g - i + \ell$ with $\ell > 0$. Let $v' = w[g\,..\,g + \ell - 1]$. We have by definition of Pref_w, $v = w[0\,..\,g - i - 1]$, $v' = w[g - i\,..\,g - i + \ell - 1]$, and $w[g + \ell] \neq w[g - i + \ell]$ (if $g + \ell < n$). So one has $w[g] = w[g - i]$ (first letter of v') and since $w[g] \neq w[g - f]$, we get $\mathrm{Pref}_w[i - f] = |v| = g - i$. The converse is also true by definition of Pref_w since $v = w[i\,..\,g - 1] = w[i - f\,..\,g - f - 1] = w[0\,..\,g - i - 1]$ (see Figure 1(iii)). ∎

The next proposition ensures that $\mathrm{Pref}_w[1\,..\,i]$ is only determined by the prefix of the word of length $\mathrm{Extent}(\mathrm{Pref}_w, i + 1)$.

Proposition 2.6. *Let $w = w[0\,..\,n - 1]$ be a word of length n and Pref_w its Prefix table. Let $i \in \{1, \ldots, n - 1\}$ and $g = \mathrm{Extent}(\mathrm{Pref}_w, i)$. Let x be the prefix of w of length g. Then $\mathrm{Pref}_w[1\,..\,i - 1] = \mathrm{Pref}_x[1\,..\,i - 1]$.*

Proof. By definition of g, we know that prefixes of w starting at j, $1 \leq j < i$, do not end beyond position $g - 1$. The letter at position g acts as a marker for all these prefixes, whence the equality. ∎

3. Testing an integer table

In this section, we describe the testing algorithm and prove its correctness.

3.1. Algorithm

The algorithm TableToWord takes as input an integer array t of size n, and checks whether t is a valid Prefix table. If so, the output is the smallest (according to the lexicographic order) word w for which $t = \mathrm{Pref}_w$ together with the number of distinct symbols needed. If t is not valid, the algorithm exits with error. It can be modified to output the longest initial part of the table that is valid.

```
TableToWord(t, n)
 1  if t[0] ≠ n then
 2        return Error("incompatible length")
 3  w[0] ← first symbol of the alphabet
 4  k ← 1
 5  g ← 1
 6  E ← ∅
 7  for i ← 1 to n − 1 do
 8        if t[i] < 0  or  t[i] > n − i then
 9              return Error("error at position", i)
10        if i = g then
11              if t[i] = 0 then
12                    w[g] ← ChooseNotIn(E ∪ {w[0]})
13                    if w[g] is a new letter then
14                          k ← k + 1
15                    E ← ∅
16                    g ← g + 1
17              elseif w[0] ∈ E then
18                    return Error("error at position", i)
19              else Copy(w, i, 0, t[i])
20                    (f, g) ← (i, i + t[i])
21                    E ← {w[g − f]}
22        elseif t[i] < g − i then                          ▷ Case (i)
23              if t[i − f] ≠ t[i] then
24                    return Error("error at position", i)
25        elseif t[i] = g − i then                          ▷ Case (ii)
26              if t[i − f] < g − i then
27                    return Error("error at position", i)
28              E ← E ∪ {w[g − i]}
29        elseif t[i − f] ≠ g − i  or  w[g − i] ∈ E then    ▷ Case (iii)
30              return Error("error at position", i)
31        else Copy(w, g, g − i, t[i] − g + i)
32              (f, g) ← (i, i + t[i])
33              E ← {w[g − f]}
34  return (w, k)
```

The function Copy is defined as follows: $\mathrm{Copy}(w, i, j, \ell)$ copies successively ℓ letters at positions $j, j+1, \ldots, j+\ell-1$ to respective positions $i, i+1, \ldots, i+\ell-1$. Note that some letters may be undefined when the process starts.

The algorithm is essentially built from the properties of Prefix tables stated in Proposition 2.5 and before the statement. The variable E implements the set $\mathcal{E}$ and stores the letters that should not appear at position g in a valid table. In the text of the algorithm,

the function ChooseNotIn(S) is used to return the first symbol which is not in the set S. Implementations of E and ChooseNotIn are discussed in Section 4.

The correctness of the algorithm is established in Propositions 3.2 and 3.3 below.

3.2. Correctness of the algorithm

Before proving that the algorithm TableToWord is correct in Propositions 3.2 and 3.3, we establish a lemma consisting in three parts. This lemma is closely related to Proposition 2.5 on prefix tables. However, the next lemma is concerned with an integer table t only known to be valid until $i-1$ and gives conditions on the next value $t[i]$.

Lemma 3.1. *In the following three statements, t is an integer table of size n, index $i \in \{1,\dots,n-1\}$ is fixed, $g = \mathrm{Extent}(t,i)$, and $\mathcal{A} = \mathrm{Anchor}(t,i)$. The table is assumed to be valid until $i-1$, compatible with the word w until $i-1$. We also suppose $i < g$. The following properties hold:*

a) *If $t[i] < g-i$ and if there exists $f \in \mathcal{A}$ such that $t[i-f] = t[i]$, then for all $f' \in \mathcal{A}$, $t[i-f'] = t[i]$.*
b) *If $t[i] = g-i$ and there exists $f \in \mathcal{A}$ such that $t[i-f] \geq g-i$, then for all $f' \in \mathcal{A}$, $t[i-f'] \geq g-i$.*
c) *If $t[i] > g-i$ and there exists $f \in \mathcal{A}$ such that $t[i-f] = g-i$ and $w[g-i] \notin \mathcal{E}(w,i,g)$, then for all $f' \in \mathcal{A}$, $t[i-f'] = g-i$.*

Proof. The proofs for the three properties rely on the same kind of arguments, but for lack of space we only detail the first one.

Let $f \in \mathcal{A}$ be such that $t[i-f] = t[i]$ and let any $f' \in \mathcal{A}$. Since t is compatible with w until $i-1$, we have both $t[f] = \mathrm{Pref}_w[f] = g-f$ and $t[f'] = \mathrm{Pref}_w[f'] = g-f'$. Let $y = w[i\,..\,i+t[i]-1]$ and $v = w[i\,..\,g-1]$. By hypothesis, y is a proper prefix of v, and the word v occurs at positions $i-f$ and $i-f'$. So, letters at positions $i+t[i]$, $i-f+t[i]$ and $i-f'+t[i]$ are the same. And this letter is different from letter at position $t[i]$ since $\mathrm{Pref}_w[i-f] = t[i]$. Therefore $t[i-f'] = \mathrm{Pref}_w[i-f'] = t[i]$. ∎

Proposition 3.2 (Correctness of the algorithm)**.** *Suppose the algorithm terminates normally (i.e., without error). At the beginning of the ith iteration,, we have the following relations for the set of variables $\{w, f, g, E\}$ of the algorithm :*

(1) $g = \mathrm{Extent}(t,i)$;
(2) *The word w built is of length $|w| = g$;*
(3) $f = \min \mathrm{Anchor}(t,i)$ *if* $i < g$;
(4) $E = \mathcal{E}(w,i,g)$;
(5) $t[1\,..\,i-1] = \mathrm{Pref}_w[1\,..\,i-1]$.

Proof. The proof is based upon recursion on the index i.

Properties 1–4 are easy to check. When $i = g$, the last property is a direct consequence both of Pref and Extent definitions and of Proposition 2.6. When $i < g$, it is a direct consequence of Lemma 3.1 as well as of propositions 2.5 and 2.6. ∎

Proposition 3.3. *If the algorithm exits with error at the ith iteration on input table t, then t is valid until position $i-1$ but not until i.*

Proof. The table is trivially invalid if the algorithm stops on line 2 or line 9. So we now suppose that $t[0] = n$ and also that integers in the table are never out of bounds.

Suppose we have an error during the ith iteration with $i > 0$. Let w be the word built so far at the beginning of the ith iteration. Thanks to Proposition 3.2, t is compatible with w until position $i-1$, so that t is also valid until position $i-1$, and we have $g = \text{Extent}(t, i)$, $f \in \text{Anchor}(t, i) = \mathcal{A}$, $E = \mathcal{E}(w, i, g)$.

We examine several cases depending on which line of the algorithm the error is produced.

Line 18. One has $i = g$, $t[i] > 0$, $E = \mathcal{E}(w, i, g) = \mathcal{E}(w, g)$ and $w[0] \in E$. There exists $f' \in \mathcal{A}$ such that $w[g - f'] = w[0]$. Thus, $t[g - f'] = \text{Pref}_w[g - f'] > 0$. By contradiction, suppose now that there exists a word z such that $t[1\,..\,i] = \text{Pref}_z[1\,..\,i]$. We must have $\text{Pref}_z[i] = t[i] > 0$ and then for all $f \in \mathcal{A}$, $z[g - f] \neq z[0]$. Thus $\text{Pref}_z[g - f] = 0$, for all $f \in \mathcal{A}$ and it is true in particular for f'. So we have $\text{Pref}_z[g - f'] = 0 \neq t[g - f']$, against the hypothesis.

Line 24 or 27. The entry $t[i]$ does not satisfies one of the properties stated in Proposition 2.5.

Line 30. We have $t[i] > g - i$. If $t[i - f] \neq g - i$, then $t[i]$ is in contradiction with Proposition 2.5 and is invalid. Suppose now that $t[i - f] = g - i$ and $w[g - i] \in E$. As in the previous case (on Line 18), there exists $f' \in \mathcal{A}$ such that $w[g - f'] = w[g - i]$ and $\text{Pref}_w[i - f'] = t[i - f'] > g - i$. Again, by contradiction, suppose that there exists a word z such that $t[1\,..\,i] = \text{Pref}_z[1\,..\,i]$. We must have $\text{Pref}_z[i] = t[i] > g - i$. By Proposition 2.5, we get $\text{Pref}_z[i - f] = g - i$, for all $f \in \mathcal{A}$, and in particular that $\text{Pref}_z[i - f'] = g - i$. Thus $\text{Pref}_z[i - f'] \neq t[i - f']$, which yields a contradiction.

As a conclusion, if the algorithm stops with error during the ith iteration, there is no word z such that $t[1\,..\,i] = \text{Pref}_z[1\,..\,i]$, which means that the table is valid until $i-1$ only. ∎

As an immediate consequence of the last two propositions we get the following statement.

Corollary 3.4. *An integer table t is a valid Prefix table if and only if the algorithm* TableToWord *terminates without error. Moreover the output word w it produces is the lexicographically smallest word compatible with t.*

4. Analysis of the algorithm

After the correctness of the algorithm TableToWord, the present section is devoted to its complexity analysis in the worst case. We first consider its running time under some assumption (proved later), show a remarkable property of contiguity of forbidden letters, and then evaluate the number of letters required to built the smallest word associated with a valid table.

4.1. Linear running time

We assume that the variable E of algorithm TableToWord is implemented with a mere linear list (linked or not). Although the variable is associated with a set of letters, we allow several copies of the same letter in the list. Therefore, adding a letter to the list or initialising it to the empty set can be performed in constant time. Membership is done in time $O(|E|)$, denoting by $|E|$ the length of the list.

In this subsection, we additionally assume that ChooseNotIn, the choice function, executes in constant time. There is no problem to get this condition if we do not require the algorithm produces the smallest word associated with a valid table. But even with

this requirement we show how to satisfy the hypothesis in Section 4.2, using a contiguity property of letters in E.

Proposition 4.1 (Time complexity)**.** *The algorithm* TableToWord *can be implemented to run in time $O(n)$ on an input integer table of size n.*

Proof. Let t be the input table. The only non constant-time instructions inside the for loop of the algorithm are copies of words (lines 19 and 31) and operations on the set of letters via the variable E (lines 6, 12, 15, 17, 21, 28, 29, and 33).

Let w be the word built by the algorithm. We note that letters of w are assigned to positions exactly once. Hence the total number of copies of letter is exactly n.

Initialisation and update operations can both be achieved in constant time with the considered implementation of E. As for membership tests at lines 17 and 29, the number of letters we have to test is at most the number of elements in $\mathrm{Anchor}(t, i)$. But we note that the value of $g = \mathrm{Extent}(t, i)$ is strictly increased eventually after the test, which means that $\mathrm{Extent}(t, i+1) > \mathrm{Extent}(t, i)$. Since by Remark 2.3 in this situation $\mathrm{Anchor}(t, i)$ is disjoint from all the $\mathrm{Anchor}(t, j)$, $j > i$, the number of letter comparisons is less than n (number of positions, excluding position 0).

It remains to consider the time to choose a letter at line 12 with function ChooseNotIn, which we assume temporarily that it is constant. We can thus conclude to an overall linear running time. ∎

4.2. Contiguous letters

We show in this section that the set of letters stored in the variable E of the algorithm TableToWord satisfies a contiguity condition. This property leads to a very simple and efficient implementation of the function ChooseNotIn, which meets the requirement stated before Proposition 4.1. We start with the extra definition of a choice position: it is a position on the table t for which the algorithm has to choose a letter.

Definition 4.2 (Choice position)**.** Let t be an integer table of size n. A position g on t is called a choice position if $\mathrm{Extent}(t, g) = g$ (that is, if for all $j < g$, $j + t[j] \leq g$) and $t[g] = 0$.

Remark. If g is a choice position, all borders of $w[0\,..\,g-1]$ (t compatible with w) are *strict borders* (if the border has length k, $w[k] \neq w[g]$). It is known that the number of strict borders of a word of length n is at most $\log_\varphi n$ where φ is the golden mean (see for example [6, Page 91]). This suggests that the alphabet size of a word admitting t as Prefix table is logarithmic in the length of t. The next section shows the logarithmic behaviour more accurately with a different argument.

Proposition 4.4 below states that each time the function ChooseNotIn is called on line 12 of the algorithm TableToWord the set $\mathcal{E}(w, g) \cup \{w[0]\}$ is a set of contiguous letters, that is, it contains all the letters between a_0 and the largest one in $\mathcal{E}(w, g) \cup \{w[0]\}$.

We introduce convenient definitions and notation related to a prefix u of the word w:

- $\mathit{Next}(u) = \{w[|v|] \mid v \in A^* \text{ border of } u\}$ (note that $\mathit{Next}(\varepsilon) = \emptyset$).
- For $a \in \mathit{Next}(u)$, the *special border* of u with respect to letter a is the shortest border v such that $w[|v|] = a$.

Lemma 4.3. *Let g be a choice position on Pref_w, and let v, $v \neq \varepsilon$, be the special border of $w[0\,..\,g-1]$ with respect to some letter b. Then $|v|$ is also a choice position.*

Proof. Since $v \neq \varepsilon$ and v is a special border, we have $b \neq w[0]$, and then $\mathrm{Pref}_w[|v|] = 0$. By contradiction, suppose $|v|$ is not a choice position. There exists a position j, $0 < j < |v|$, such that $j + \mathrm{Pref}_w[j] > |v|$. Let $y = w[j \,..\, |v|-1]$, which is a border of v, and also a border of $w[0 \,..\, g-1]$. But we also have $b = w[|y|]$ and $|y| < |v|$. So v cannot be the special border with respect to b. ∎

Proposition 4.4 (Contiguous)**.** *Let t be a valid Prefix table and w the word produced by the algorithm* TableToWord *from t. For any choice position g on t, setting $k = \mathrm{card}\, Next(w[0 \,..\, g-1])$, we have $Next(w[0 \ldots g-1]) = \{a_0, \ldots, a_{k-1}\}$.*

Proof. Note that choice positions are exactly positions where we must call the function ChooseNotIn (line 12). In this case we always choose the smallest possible letter.

The proof is based upon a recursion on k. In the following g is always a choice position and we set $S_g = Next(w[0 \,..\, g-1])$ and $w_g = w[0 \,..\, g-1]$ to shorten notation.

If $k = 1$, we have $S_g = \{w[0]\}$ since ε is always a border of w_g.
Suppose that for any choice position h with $\mathrm{card}(S_h) = k-1$, we have $S_h = \{a_0, a_1, \ldots, a_{k-2}\}$. Let g be a choice position such that $\mathrm{card}(S_g) = k$ with $k > 1$. Consider the longest special border u of w_g and let $b = w[|u|]$ be its associated letter. We have $u \neq \varepsilon$ since $\mathrm{card}(S_g) > 1$. The previous lemma entails that $|u|$ is a choice position. But b cannot be in $S_{|u|}$ (otherwise u would not be the special border with respect to b). Since $S_g = S_{|u|} \cup \{b\}$, we must have $\mathrm{card}(S_{|u|}) = k-1$. So by recurrence $S_{|u|} = \{a_0, a_1, \ldots, a_{k-2}\}$. Therefore the letter chosen at position $|u|$ is $b = a_{k-1}$.

This ends the recurrence and the proof. ∎

Implementation of ChooseNotIn. Following Proposition 4.4, a simple way to implement the function ChooseNotIn is to store the largest forbidden letter at position g. Choosing a letter remains to take the next letter because of the contiguity property, which can obviously be done in constant time.

4.3. Alphabet size

We evaluate the smallest number of letters needed to build a word associated with a valid Prefix table. We first adapt some properties on borders of a word derived from results by Hancart [16] (see also [6] or [19] for example). In the following, w is a word of length n over the alphabet A.

For any prefix of u of w, we define

$$deg(u) = \mathrm{card}(Next(u)),$$

and note that it is $\mathrm{card}(\mathcal{E}(w, |u|) \cup \{w[0]\})$ when $u \neq \varepsilon$. We have

$$deg(u) = \begin{cases} 0 & \text{if } u = \varepsilon \\ \mathrm{card}\{v \mid v \text{ is a special border of } u\} & \text{otherwise.} \end{cases}$$

Lemma 4.5. *Let u be a nonempty prefix of w. Any special border v of u satisfies the inequality $|v| < |u|/2$.*

Proof. The word v is a special border with respect to $a = w[|v|]$. We prove the statement by contradiction. Suppose $|v| \geq |u|/2$, and let $k = 2|v| - |u|$ and $y = w[0 \,..\, k-1]$. Then y is a border of v (and also of u, see Figure 2) and $|y| < |v|$. But the equality $w[|y|]$ and $w[|v|]$ contradicts the fact that v is a special border. Thus, the inequality holds. ∎

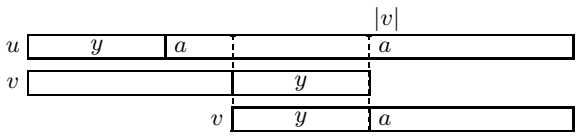


Figure 2: The word y is a border of v and u. Letters at positions $|y|$ and $|v|$ are the same, which prevents v to be a special border of u.

Denoting by $s\text{-}Bord(u)$ the longest special border of u, we get the following corollary.

Corollary 4.6. *Let $w \in A^+$. The length of $s\text{-}Bord(w)$ is smaller than $|w|/2$.*

Lemma 4.7. *Let $w \in A^+$ and u a nonempty prefix of w. One has*

$$deg(u) = deg(s\text{-}Bord(u)) + 1.$$

Proof. This is a direct consequence of the fact that a special border of u is either $s\text{-}Bord(u)$ or a special border of $s\text{-}Bord(u)$. ∎

Proposition 4.8. *For any nonempty prefix u of w, we have*

$$deg(u) \leq \lfloor \log_2(|u| + 1) \rfloor.$$

Proof. The proof relies on a recursion on $|u|$. If $|u| = 1$, then $deg(u) = 1$ since ε is the unique border of u. Thus the property is true. Suppose $|u| > 1$ and that the property holds for any prefix of w of length less than $|u|$. Let us set $v = s\text{-}Bord(u)$. We have

$$deg(u) = deg(v) + 1 \leq \lfloor \log_2(|v| + 1) \rfloor + 1 = \lfloor \log_2(2|v| + 2) \rfloor \leq \lfloor \log_2(|u| + 1) \rfloor.$$

The first equality comes from Lemma 4.7, and the last inequality is obtained *via* Corollary 4.6. ∎

The last lemma yields the following corollary.

Corollary 4.9. *For any nonempty prefix u of w, one has*

$$deg(u) \leq \min\{\mathrm{card}(Alpha(u)), \lfloor \log_2(|u| + 1) \rfloor\},$$

where $Alpha(u)$ is the set of letters occurring in u.

The following example shows that the bound stated in Corollary 4.9 is tight.

Example. We define a sequence of word $(w_i)_{i \geq 0}$ by

$$w_0 = a_0, \text{ and for } i \geq 1 \; w_i = w_{i-1} a_i w_{i-1}.$$

It is straightforward to check that, for $i \geq 0$, we have both $|w_i| = 2^{i+1} - 1$ and $deg(w_i) = \mathrm{card}(Alpha(w_i)) = \lfloor \log_2(|w_i| + 1) \rfloor = i + 1$.

Proposition 4.10. *For the word w produced by the algorithm* TableToWord *from a table of size n, the following (tight) inequality holds*

$$\mathrm{card}(Alpha(w)) \leq \lfloor \log_2(n + 1) \rfloor.$$

Proof. With the same notation as in Proposition 3.2, at the beginning of the ith iteration, the word w of length $g = \mathrm{Extent}(t, i)$ has been built, and we have $E = \mathcal{E}(w, i, g)$. So if $i = g$ and $t[i] = 0$, we have $E \cup \{w[0]\} = Next(w)$. Hence, by Corollary 4.9, we get

$$\mathrm{card}(E \cup \{w[0]\}) = deg(w) \leq \min\{\mathrm{card}(Alpha(w)), \lfloor \log_2(|w| + 1) \rfloor\}.$$

This implies that a new letter must be introduced only if $\mathrm{card}(E \cup \{w[0]\}) = \mathrm{card}(Alpha(w))$. Considering $i = n$ yields the result. ∎

Remark (Size of the maximal alphabet). It is worth noting that a slight modification of the algorithm provides a word with the maximal number of letters and compatible with the input valid table. Indeed we just have, at each call to the function ChooseNotIn, to return a new letter each time it is called. This way we build the word compatible with t with the highest number of letters.

Acknowledgements. We are grateful to Pat Ryan, Bill Smyth, and Shu Wang for interesting discussions on Prefix tables and their extension to wider problems. We thank reviewers for their careful reading of the article and appropriate remarks.

References

[1] A. Apostolico and R. Giancarlo. The Boyer-Moore-Galil string searching strategies revisited. *SIAM J. Comput.*, 15(1):98–105, 1986.

[2] R. Baeza-Yates, C. Choffrut, and G. Gonnet. On Boyer-Moore automata. *Algorithmica*, 12(4/5):268–292, 1994.

[3] H. Bannai, S. Inenaga, A. Shinohara, and M. Takeda. Inferring strings from graphs and arrays. In B. Rovan and P. Vojtás, editors, *Mathematical Foundations of Computer Science*, volume 2747 of *Lecture Notes in Computer Science*, pages 208–217. Springer, 2003.

[4] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[5] M. Crochemore, J. Désarménien, and D. Perrin. A note on the Burrows-Wheeler transformation. *Theoretical Computer Science*, 332(1-3):567–572, 2005.

[6] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on Strings*. Cambridge University Press, Cambridge, UK, 2007. 392 pages.

[7] M. Crochemore and L. Ilie. Computing Longest Previous Factor in linear time and applications. *Inf. Process. Lett.*, 106(2):75–80, 2008.

[8] M. Crochemore, L. Ilie, and W. F. Smyth. A simple algorithm for computing the Lempel-Ziv factorization. In J. A. Storer and M. W. Marcellin, editors, *18th Data Compression Conference*, pages 482–488. IEEE Computer Society, Los Alamitos, CA, 2008.

[9] M. Crochemore and T. Lecroq. Tight bounds on the complexity of the Apostolico-Giancarlo algorithm. *Information Processing Letters*, 63(4):195–203, 1997.

[10] J.-P. Duval, T. Lecroq, and A. Lefebvre. Border array on bounded alphabet. *Journal of Automata, Languages and Combinatorics*, 10(1):51–60, 2005.

[11] J.-P. Duval, T. Lecroq, and A. Lefebvre. Efficient validation and construction of border arrays. In *Proceedings of 11th Mons Days of Theoretical Computer Science*, pages 179–189, Rennes, France, 2006.

[12] F. Franek, S. Gao, W. Lu, P. J. Ryan, W. F. Smyth, Y. Sun, and L. Yang. Verifying a Border array in linear time. *J. Combinatorial Math. and Combinatorial Computing*, 42:223–236, 2002.

[13] F. Franek and W. F. Smyth. Reconstructing a Suffix Array. *J. Foundations of Computer Sci.*, 17(6):1281–1295, 2006.

[14] Z. Galil and J. I. Seiferas. A linear-time on-line recognition algorithm for "palstar". *J. ACM*, 25(1):102–111, 1978.

[15] D. Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, Cambridge, UK, 1997. 534 pages.

[16] C. Hancart. On Simon's string searching algorithm. *Inf. Process. Lett.*, 47(2):95–99, 1993.

[17] D. E. Knuth, J. H. M. Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(1):323–350, 1977.

[18] G. Manacher. A new linear-time on-line algorithm finding the smallest initial palindrome of a string. *J. Assoc. Comput. Mach.*, 22(3):346–351, 1975.

[19] W. F. Smyth. *Computing Patterns in Strings*. Pearson, Essex, UK, 2003.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 301–312
www.stacs-conf.org

THE PRICE OF ANARCHY IN COOPERATIVE NETWORK CREATION GAMES

ERIK D. DEMAINE [1] AND MOHAMMADTAGHI HAJIAGHAYI [1,2] AND HAMID MAHINI [3,4] AND MORTEZA ZADIMOGHADDAM [4]

[1] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA
E-mail address: edemaine@mit.edu

[2] AT&T Labs — Research, 180 Park Ave., Florham Park, NJ 07932, USA
E-mail address: hajiagha@research.att.com

[3] School of Computer Science, Institute for Theoretical Physics and Mathematics, Tehran
E-mail address: mahini@ce.sharif.edu

[4] Department of Computer Engineering, Sharif University of Technology
E-mail address: zadimoghaddam@ce.sharif.edu

Abstract. We analyze the structure of equilibria and the price of anarchy in the family of network creation games considered extensively in the past few years, which attempt to unify the network design and network routing problems by modeling both creation and usage costs. In general, the games are played on a host graph, where each node is a selfish independent agent (player) and each edge has a fixed link creation cost α. Together the agents create a network (a subgraph of the host graph) while selfishly minimizing the link creation costs plus the sum of the distances to all other players (usage cost). In this paper, we pursue two important facets of the network creation game.

First, we study extensively a natural version of the game, called the cooperative model, where nodes can collaborate and share the cost of creating any edge in the host graph. We prove the first nontrivial bounds in this model, establishing that the price of anarchy is polylogarithmic in n for all values of α in complete host graphs. This bound is the first result of this type for any version of the network creation game; most previous general upper bounds are polynomial in n. Interestingly, we also show that equilibrium graphs have polylogarithmic diameter for the most natural range of α (at most $n \operatorname{polylg} n$).

Second, we study the impact of the natural assumption that the host graph is a general graph, not necessarily complete. This model is a simple example of nonuniform creation costs among the edges (effectively allowing weights of α and ∞). We prove the first assemblage of upper and lower bounds for this context, establishing nontrivial tight bounds for many ranges of α, for both the unilateral and cooperative versions of network creation. In particular, we establish polynomial lower bounds for both versions and many ranges of α, even for this simple nonuniform cost model, which sharply contrasts the conjectured constant bounds for these games in complete (uniform) graphs.

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 301-312
http://drops.dagstuhl.de/opus/volltexte/2009/1839

1. Introduction

A fundamental family of problems at the intersection between computer science and operations research is *network design*. This area of research has become increasingly important given the continued growth of computer networks such as the Internet. Traditionally, we want to find a minimum-cost (sub)network that satisfies some specified property such as k-connectivity or connectivity on terminals (as in the classic Steiner tree problem). This goal captures the (possibly incremental) creation cost of the network, but does not incorporate the cost of actually using the network. In contrast, *network routing* has the goal of optimizing the usage cost of the network, but assumes that the network has already been created.

Network creation games attempt to unify the network design and network routing problems by modeling both creation and usage costs. In general, the game is played on a *host graph*, where each node is an independent agent (player), and the goal is to create a network from a subgraph of the host graph. Collectively, the nodes decide which edges of the host graph are worth creating as links in the network. Every link has the same creation cost α. (Equivalently, links have creation costs of α and ∞, depending on whether they are edges of the host graph.) In addition to these creation costs, each node incurs a usage cost equal to the sum of distances to all other nodes in the network. Equivalently, if we divide the cost (and thus α) by the number n of nodes, the usage cost for each node is its average distance to all other nodes. (This natural cost model has been used in, e.g., contribution games and network-formation games.)

There are several versions of the network creation game that vary how links are purchased. In the *unilateral* model—introduced by Fabrikant, Luthra, Maneva, Papadimitriou, and Shenker [15]—every node (player) can locally decide to purchase any edge incident to the node in the host graph, at a cost of α. In the *bilateral* model—introduced by Corbo and Parkes [9]—both endpoints of an edge must agree before they can create a link between them, and the two nodes share the α creation cost equally. In the *cooperative* model—introduced by Albers, Eilts, Even-Dar, Mansour, and Roditty [2]—any node can purchase any amount of any edge in the host graph, and a link gets created when the total purchased amount is at least α.

To model the dominant behavior of large-scale networking scenarios such as the Internet, we consider the case where every node (player) selfishly tries to minimize its own creation and usage cost [18, 15, 2, 9]. This game-theoretic setting naturally leads to the various kinds of *equilibria* and the study of their structure. Two frequently considered notions are *Nash equilibrium* [24, 25], where no player can change its strategy (which edges to buy) to locally improve its cost, and *strong Nash equilibrium* [6, 3, 1], where no coalition of players can change their collective strategy to locally improve the cost of each player in the coalition. Nash equilibria capture the combined effect of both selfishness and lack of coordination, while strong Nash equilibria separates these issues, enabling coordination and capturing the specific effect of selfishness. However, the notion of strong Nash equilibrium is extremely restrictive in our context, because all players can simultaneously change their entire strategies, abusing the local optimality intended by original Nash equilibria, and effectively forcing globally near-optimal solutions [3].

We consider weaker notions of equilibria, which broadens the scope of equilibria and therefore strengthens our upper bounds, where players can change their strategy on only a single edge at a time. In a *collaborative equilibrium*, even coalitions of players do not wish to

change their collective strategy on any single edge; this concept is particularly important for the cooperative network creation game, where multiple players must negotiate their relative valuations of an edge. (This notion is the natural generalization of pairwise stability from [9] to arbitrary cost sharing.) Collaborative equilibria are essentially a compromise between Nash and strong Nash equilibria: they still enable coordination among players and thus capture the specific effect of selfishness, like strong Nash, yet they consider more local moves, in the spirit of Nash. In particular, any results about all collaborative equilibria also apply to all strong Nash equilibria. Collaborative equilibria also make more sense computationally: players can efficiently detect equilibrium using a simple bidding procedure (whereas this problem is NP-hard for strong Nash), and the resulting dynamics converge to such equilibria (see Section 2.2).

The structure of equilibria in network creation games is not very well understood. For example, Fabrikant et al. [15] conjectured that equilibrium graphs in the unilateral model were all trees, but this conjecture was disproved by Albers et al. [2]. One particularly interesting structural feature is whether all equilibrium graphs have small *diameter* (say, polylogarithmic), analogous to the small-world phenomenon [19, 14], In the original unilateral version of the problem, the best general lower bound is just a constant and the best general upper bound is polynomial. A closely related issue is the *price of anarchy* [20, 26, 28], that is, the worst possible ratio of the total cost of an equilibrium (found by independent selfish behavior) and the optimal total cost possible by a centralized solution (maximizing social welfare). The price of anarchy is a well-studied concept in algorithmic game theory for problems such as load balancing, routing, and network design; see, e.g., [26, 10, 27, 15, 5, 4, 8, 9, 2, 11]. Upper bounds on diameter of equilibrium graphs translate to approximately equal upper bounds on the price of anarchy, but not necessarily vice versa. In the unilateral version, for example, there is a general $2^{O(\sqrt{\lg n})}$ upper bound on the price of anarchy.

Previous work. Network creation games have been studied extensively in the literature since their introduction in 2003.

For the unilateral version and a complete host graph, Fabrikant et al. [15] prove an upper bound of $O(\sqrt{\alpha})$ on the price of anarchy for all α. Lin [23] proves that the price of anarchy is constant for two ranges of α: $\alpha = O(\sqrt{n})$ and $\alpha \geq c\,n^{3/2}$ for some $c > 0$. Independently, Albers et al. [2] prove that the price of anarchy is constant for $\alpha = O(\sqrt{n})$, as well as for the larger range $\alpha \geq 12\,n\lceil \lg n \rceil$. In addition, Albers et al. prove a general upper bound of $15\left(1 + (\min\{\frac{\alpha^2}{n}, \frac{n^2}{\alpha}\})^{1/3}\right)$. The latter bound shows the first sublinear worst-case bound, $O(n^{1/3})$, for all α. Demaine et al. [11] prove the first $o(n^{\varepsilon})$ upper bound for general α, namely, $2^{O(\sqrt{\lg n})}$. They also prove a constant upper bound for $\alpha = O(n^{1-\varepsilon})$ for any fixed $\varepsilon > 0$, and improve the constant upper bound by Albers et al. (with the lead constant of 15) to 6 for $\alpha < (n/2)^{1/2}$ and to 4 for $\alpha < (n/2)^{1/3}$. Andelmen et al. [3] show that, among strong Nash equilibria, the price of anarchy is at most 2.

For the bilateral version and a complete host graph, Corbo and Parkes [9] prove that the price of anarchy is between $\Omega(\lg \alpha)$ and $O(\min\{\sqrt{\alpha}, n/\sqrt{\alpha})$. Demaine et al. [11] prove that the upper bound is tight, establishing the price of anarchy to be $\Theta(\min\{\sqrt{\alpha}, n/\sqrt{\alpha}\})$ in this case.

For the cooperative version and a complete host graph, the only known result is an upper bound of $15\left(1 + (\min\{\frac{\alpha^2}{n}, \frac{n^2}{\alpha}\})^{1/3}\right)$, proved by Albers et al. [2].

Other variations of network creation games allow nonuniform interests in connectivity between nodes [17] and nodes with limited budgets for buying edges [21].

Our results. Our research pursues two important facets of the network creation game.

First, we make an extensive study of a natural version of the game—the cooperative model—where the only previous results were simple extensions from unilateral analysis. We substantially improve the bounds in this case, showing that the price of anarchy is polylogarithmic in n for *all values of* α in complete graphs. This is the first result of this type for any version of the network creation game. As mentioned above, this result applies to both collaborative equilibria and strong Nash equilibria. Interestingly, we also show that equilibrium graphs have polylogarithmic diameter for the most natural range of α (at most $n \operatorname{polylg} n$). Note that, because of the locally greedy nature of Nash equilibria, we cannot use the classic probabilistic spanning (sub)tree embedding machinery of [7, 16, 13] to obtain polylogarithmic bounds (although this machinery can be applied to approximate the global social optimum).

Second, we study the impact of the natural assumption that the host graph is a general graph, not necessarily complete, inspired by practical limitations in constructing network links. This model is a simple example of nonuniform creation costs among the edges (effectively allowing weights of α and ∞). Surprisingly, no bounds on the diameter or the price of anarchy have been proved before in this context. We prove several upper and lower bounds, establishing nontrivial tight bounds for many ranges of α, for both the unilateral and cooperative versions. In particular, we establish polynomial lower bounds for both versions and many ranges of α, even for this simple nonuniform cost model. These results are particularly interesting because, by contrast, no superconstant lower bound has been shown for either game in complete (uniform) graphs. Thus, while we believe that the price of anarchy is polylogarithmic (or even constant) for complete graphs, we show a significant departure from this behavior in general graphs.

Our proof techniques are most closely related in spirit to "region growing" from approximation algorithms; see, e.g., [22]. Our general goal is to prove an upper bound on diameter by way of an upper bound on the expansion of the graph. However, we have not been able to get such an argument to work directly in general. The main difficulty is that, if we imagine building a breadth-first-search tree from a node, then connecting that root node to another node does not necessarily benefit the node much: it may only get closer to a small fraction of nodes in the BFS subtree. Thus, no node is motivated selfishly to improve the network, so several nodes must coordinate their changes to make improvements. The cooperative version of the game gives us some leverage to address this difficulty. We hope that this approach, particularly the structure we prove of equilibria, will shed some light on the still-open unilateral version of the game, where the best bounds on the price of anarchy are $\Omega(1)$ and $2^{O(\sqrt{\lg n})}$.

Table 1 summarizes our results. Section 4 proves our polylogarithmic upper bounds on the price of anarchy for all ranges of α in the cooperative network creation game in complete graphs. Section 5 considers how the cooperative network creation game differs in general graphs, and proves our upper bounds for this model. Section 6 extends these results to apply to the unilateral network creation game in general graphs. Section 7 proves lower bounds for both the unilateral and cooperative network creation games in general graphs, which match our upper bounds for some ranges of α.

$\alpha =$	0	n	$n \lg^{0.52} n$	$n \lg^{7.16} n$	$n^{3/2}$	$n^{5/3}$	n^2	$n^2 \lg n$	∞
Cooperative, complete graph	$\Theta(1)$	$\lg^{3.32} n$	$O\big(\lg n + \sqrt{\frac{n}{\alpha}} \lg^{3.58} n\big)$	$\Theta(1)$					
Cooperative, general graph	$O(\alpha^{1/3})$	$O(n^{1/3})$, $\Omega(\sqrt{\frac{\alpha}{n}})$				$\Theta(\frac{n^2}{\alpha})$	$O\big(\frac{n^2}{\alpha} \lg n\big)$	$\Theta(1)$	
Unilateral, general graph	$O(\alpha^{1/2})$	$O(n^{1/2})$, $\Omega(\frac{\alpha}{n})$			$\Theta(\frac{n^2}{\alpha})$		$\Theta(1)$		

Table 1: Summary of our bounds on equilibrium diameter and price of anarchy for cooperative network creation in complete graphs, and unilateral and cooperative network creation in general graphs. For all three of these models, our bounds are strict improvements over the best previous bounds.

2. Models

In this section, we formally define the different models of the network creation game.

2.1. Unilateral Model

We start with the unilateral model, introduced in [15]. The game is played on a *host graph* $G = (V, E)$. Assume $V = \{1, 2, \ldots, n\}$. We have n players, one per vertex. The strategy of player i is specified by a subset s_i of $\{j : \{i, j\} \in E\}$, defining the set of neighbors to which player i creates a link. Thus each player can only create links corresponding to edges incident to node i in the host graph G Together, let $s = \langle s_1, s_2, \ldots, s_n \rangle$ denote the joint strategy of all players.

To define the cost of strategies, we introduce a spanning subgraph G_s of the host graph G. Namely, G_s has an edge $\{i, j\} \in E(G)$ if either $i \in s_j$ or $j \in s_i$. Define $d_{G_s}(i, j)$ to be the distance between vertices i and j in graph G_s. Then the cost incurred by player i is $c_i(s) = \alpha\,|s_i| + \sum_{j=1}^{n} d_{G_s}(i, j)$. The total cost incurred by joint strategy s is $c(s) = \sum_{i=1}^{n} c_i(s)$.

A (pure) *Nash equilibrium* is a joint strategy s such that $c_i(s) \leq c_i(s')$ for all joint strategies s' that differ from s in only one player i. The *price of anarchy* is then the maximum cost of a Nash equilibrium divided by the minimum cost of any joint strategy (called the *social optimum*).

2.2. Cooperative Model

Next we turn to the cooperative model, introduced in [15, 2]. Again, the game is played on a host graph $G = (V, E)$, with one player per vertex. Assume $V = \{1, 2, \ldots, n\}$ and $E = \{e_1, e_2, \ldots, e_{|E|}\}$. Now the strategy of player i is specified by a vector $s_i = \langle s(i, e_1), s(i, e_2), \ldots, s(i, e_{|E|}) \rangle$, where $s(i, e_j)$ corresponds to the value that player i is willing to pay for link e_j. Together, $s = \langle s_1, s_2, \ldots, s_n \rangle$ denotes the strategies of all players.

We define a spanning subgraph $G_s = (V, E_s)$ of the host graph G: e_j is an edge of G_s if $\sum_{i \in V(G)} s(i, e_j) \geq \alpha$. To make the total cost for an edge e_j exactly 0 or α in all cases, if $\sum_{i \in V(G)} s(i, e_j) > \alpha$, we uniformly scale the costs to sum to α: $s'(i, e_j) = \alpha s(i, e_j) / \sum_{k \in V(G)} s(k, e_j)$ (Equilibria will always have $s = s'$.) Then the cost incurred by player i is $c_i(s) = \sum_{e_j \in E_s} s'(i, e_j) + \sum_{j=1}^{n} d_{G_s}(i, j)$. The total cost incurred by joint strategy s is $c(s) = \alpha\,|E_s| + \sum_{i=1}^{n} \sum_{j=1}^{n} d_{G_s}(i, j)$.

In this cooperative model, the notion of Nash equilibrium is less natural because it allows only one player to change strategy, whereas a cooperative purchase in general requires many players to change their strategy. Therefore we use a stronger notion of equilibrium that allows coalition among players, inspired by the strong Nash equilibrium of Aumann

[6], and modeled after the pairwise stability property introduced for the bilateral game [9]. Namely, a joint strategy s is *collaboratively equilibrium* if, for any edge e of the host graph G, for any coalition $C \subseteq V$, for any joint strategy s' differing from s in only $s'(i,e)$ for $i \in C$, some player $i \in C$ has $c_i(s') > c_i(s)$. Note that any such joint strategy must have every sum $\sum_{i \in V(G)} s(i,e_j)$ equal to either 0 or α, so we can measure the cost $c_i(s)$ in terms of $s(i,e_j)$ instead of $s'(i,e_j)$. The *price of anarchy* is the maximum cost of a collaborative equilibrium divided by the minimum cost of any joint strategy (the *social optimum*).

We can define a simple dynamics for the cooperative network creation game in which we repeatedly pick a pair of vertices, have all players determine their valuation of an edge between those vertices (change in $c_i(s)$ from addition or removal), and players thereby bid on the edge and change their strategies. These dynamics always converge to a collaborative equilibrium because each change decreases the total cost $c(s)$, which is a discrete quantity in the lattice $\mathbb{Z} + \alpha\mathbb{Z}$. Indeed, the system therefore converges after a number of steps polynomial in n and the smallest integer multiple of α (if one exists). More generally, we can show an exponential upper bound in terms of just n by observing that the graph uniquely determines $c(s)$, so we can never repeat a graph by decreasing $c(s)$.

3. Preliminaries

In this section, we define some helpful notation and prove some basic results. Call a graph G_s corresponding to an equilibrium joint strategy s an *equilibrium graph*. In such a graph, let $d_{G_s}(u,v)$ be the length of the shortest path from u to v and $\mathrm{Dist}_{G_s}(u)$ be $\sum_{v \in V(G_s)} d_{G_s}(u,v)$. Let $N_k(u)$ denote the set of vertices with distance at most k from vertex u, and let $N_k = \min_{v \in G} |N_k(v)|$. In both the unilateral and cooperative network creation games, the total cost of a strategy consists of two parts. We refer to the cost of buying edges as the *creation cost* and the cost $\sum_{v \in V(G_s)} d_{G_s}(u,v)$ as the *usage cost*.

First we prove the existence of collaborative equilibria for complete host graphs. Similar results are known in the unilateral case [15, 3].

Lemma 3.1. *In the cooperative network creation game, any complete graph is a collaborative equilibrium for $\alpha \leq 2$, and any star graph is a collaborative equilibrium for $\alpha \geq 2$.*

Next we show that, in the unilateral version, a bound on the usage cost suffices to bound the total cost of an equilibrium graph G_s, similar to [11, Lemma 1].

Lemma 3.2. *The total cost of any equilibrium graph in the unilateral game is at most $\alpha\, n + 2 \sum_{u,v \in V(G_s)} d_{G_s}(u,v)$.*

Next we prove a more specific bound for the cooperative version, using the following bound on the number of edges in a graph of large girth:

Lemma 3.3. [12] *The number of edges in an n-vertex graph of odd girth g is $O(n^{1+2/(g-1)})$.*

Lemma 3.4. *For any integer g, the total cost of any equilibrium graph G_s is at most $\alpha\, O(n^{1+2/g}) + g \sum_{u,v \in V(G_s)} d_{G_s}(u,v)$.*

4. Cooperative Version in Complete Graphs

In this section, we study the price of anarchy when any number of players can cooperate to create any link, and the host graph is the complete graph.

We start with two lemmata that hold for both the unilateral and cooperative versions of the problem. The first lemma bounds a kind of "doubling radius" of large neighborhoods around any vertex, which the second lemma uses to bound the usage cost.

Lemma 4.1. [11, Lemma 4] *For any vertex u in an equilibrium graph G_s, if $|N_k(u)| > n/2$, then $|N_{2k+2\alpha/n}(u)| \geq n$.*

Lemma 4.2. *If we have $N_k(u) > n/2$ for some vertex u in an equilibrium graph G_s, the usage cost is at most $O(n^2k + \alpha n)$.*

Next we show how to improve the bound on "doubling radius" for large neighborhoods in the cooperative game:

Lemma 4.3. *For any vertex u in an equilibrium graph G_s, if $|N_k(u)| > n/2$, then $|N_{2k+4\sqrt{\alpha/n}}(u)| \geq n$.*

Next we consider what happens with arbitrary neighborhoods, using techniques similar to [11, Lemma 5].

Lemma 4.4. *If $|N_k(u)| \geq Y$ for every vertex u in an equilibrium graph G_s, then either $|N_{4k+2}(u)| > n/2$ for some vertex u or $|N_{5k+3}(u)| \geq Y^2 n/\alpha$ for every vertex u.*

Proof. If there is a vertex u with $|N_{4k+2}(u)| > n/2$, then the claim is obvious. Otherwise, for every vertex u, $|N_{4k+2}(u)| \leq n/2$. Let u be an arbitrary vertex. Let S be the set of vertices whose distance from u is $4k + 3$. We select a subset of S, called *center points*, by the following greedy algorithm. We repeatedly select an unmarked vertex $z \in S$ as a center point, mark all unmarked vertices in S whose distance from z is at most $2k$, and assign these vertices to z.

Suppose that we select l vertices $x_1, x_2, \ldots, x_l$ as center points. We prove that $l \geq |N_k(u)|n/\alpha$. Let C_i be the vertices in S assigned to x_i; see Figure 1. By construction, $S = \bigcup_{i=1}^{l} C_i$. We also assign each vertex v at distance at least $4k+4$ from u to one of these center points, as follows. Pick any one shortest path from v to u that contains some vertex $w \in S$, and assign v to the same center point as w. This vertex w is unique in this path because this path is a shortest path from v to u. Let T_i be the set of vertices assigned to x_i and whose distance from u is more than $4k + 2$. By construction, $\bigcup_{i=1}^{l} T_i$ is the set of vertices at distance more than $4k + 2$ from u. The shortest path from $v \in T_i$ to u uses some vertex $w \in C_i$. For any vertex x whose distance is at most k from u and for any $y \in T_i$, adding the edge $\{u, x_i\}$ decreases the distance

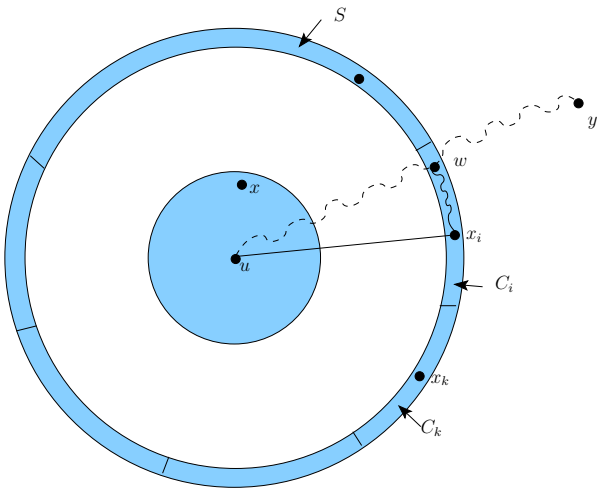


Figure 1: Center points.

between x and y at least 2, because the shortest path from $y \in T_i$ to u uses some vertex $w \in C_i$, as shown in Figure 1. By adding edge $\{u, x_i\}$, the distance between u and w would become at most $2k+1$ and the distance between x and w would become at most $3k+1$, where x is any vertex whose distance from u is at most k. Because the current distance between x and w is at least $4k+3-k = 3k+3$, adding the edge $\{u, x_i\}$ decreases this distance by at least 2. Consequently the distance between x and any $y \in T_i$ decreases by at least 2. Note that the distance between x and y is at least $d_{G_s}(u, y) - k$, and after adding edge (u, x_i), this distance becomes at most $3k+1+d_{G_s}(w, y) = 3k+1+d_{G_s}(u, y) - d_{G_s}(u, w) = 3k+1+d_{G_s}(u, y) - (4k+3) = d_{G_s}(u, y) - k - 2$.

Thus any vertex $y \in T_i$ has incentive to pay at least $2\,|N_k(u)|$ for edge $\{u, x_i\}$. Because the edge $\{u, x_i\}$ is not in equilibrium, we conclude that $\alpha \geq 2|T_i||N_k(u)|$. On the other hand, $|N_{4k+2}(u)| \leq n/2$, so $\sum_{i=1}^{l} |T_i| \geq n/2$. Therefore, $l\,\alpha \geq 2|N_k(u)| \sum_{i=1}^{l} |T_i| \geq n|N_k(u)|$ and hence $l \geq n|N_k(u)|/\alpha$.

According to the greedy algorithm, the distance between any pair of center points is more than $2k$; hence, $N_k(x_i) \cap N_k(x_j) = \emptyset$ for $i \neq j$. By the hypothesis of the lemma, $|N_k(x_i)| \geq Y$ for every vertex x_i; hence $|\bigcup_{i=1}^{l} N_k(x_i)| = \sum_{i=1}^{l} |N_k(x_i)| \geq l\,Y$. For every $i \leq l$, we have $d_{G_s}(u, x_i) = 4k+3$, so vertex u has a path of length at most $5k+3$ to every vertex whose distance to x_i is at most k. Therefore, $|N_{5k+3}(u)| \geq |\bigcup_{i=1}^{l} N_k(x_i)| \geq l\,Y \geq Yn|N_k(u)|/\alpha \geq Y^2 n/\alpha$. □

Now we are ready to prove bounds on the price of anarchy. We start with the case when α is a bit smaller than n:

Theorem 4.5. *For $1 \leq \alpha < n^{1-\varepsilon}$, the price of anarchy is at most $O(1/\varepsilon^{1+\lg 5})$.*

Next we prove a polylogarithmic bound on the price of anarchy when α is close to n.

Theorem 4.6. *For $\alpha = O(n)$, the price of anarchy is $O(\lg^{1+\lg 5} n)$ and the diameter of any equilibrium graph is $O(\lg^{\lg 5} n)$.*

Proof. Consider an equilibrium graph G_s. The proof is similar to the proof of Theorem 4.5. Define $a_1 = \max\{2, 2\alpha/n\} + 1$ and $a_i = 5a_{i-1} + 3$, or equivalently $a_i = \frac{4a_1+3}{20} \cdot 5^i - \frac{3}{4} < a_1 5^i$, for all $i > 1$. By Lemma 4.4, for each $i \geq 1$, either $N_{4a_i+2}(v) > n/2$ for some vertex v or $N_{a_{i+1}} \geq (n/\alpha)\, N_{a_i}^2$. Let j be the least number for which $|N_{4a_j+2}(v)| > n/2$ for some vertex v. By this definition, for each $i < j$, $N_{a_{i+1}} \geq (n/\alpha)\, N_{a_i}^2$. Because $N_{a_1} > 2\max\{1, \alpha/n\}$, we obtain that $N_{a_i} > 2^{2^{i-1}} \max\{1, \alpha/n\}$ for every $i \leq j$. On the other hand, $2^{2^{j-1}} \leq 2^{2^{j-1}} \max\{1, \alpha/n\} < N_{a_j} \leq n$, so $j < \lg\lg n + 1$ and $a_j < a_1\, 5^{\lg\lg n+1} < (2 + 2\alpha/n + 1 + 1)5 \lg^{\lg 5} n = 10(2 + \alpha/n) \lg^{\lg 5} n$. Therefore $N_{4\cdot[10(2+\alpha/n)\lg^{\lg 5} n]+2}(v) > n/2$ for some vertex v and using Lemma 4.1, we conclude that the distance of v to all other vertices is at most $2[40(2 + \alpha/n) \lg^{\lg 5} n + 2] + 2\alpha/n$. Thus the diameter of G_s is at most $O((1 + \alpha/n) \lg^{\lg 5} n)$. Setting $g = \lg n$ in Lemma 3.4, the cost of G_s is at most $\alpha\, O(n) + (\lg n) O(n^2(1 + \alpha/n) \lg^{\lg 5} n) = O((\alpha n + n^2) \lg^{1+\lg 5} n)$. Therefore the price of anarchy is at most $O(\lg^{1+\lg 5} n)$. □

When α is a bit larger than n, we can obtain a constant bound on the price of anarchy. First we need a somewhat stronger result on the behavior of neighborhoods:

Lemma 4.7. *If $|N_k(u)| \geq Y$ for every vertex u in an equilibrium graph G_s, then either $|N_{5k}(u)| > n/2$ for some vertex u or $|N_{6k+1}(u)| \geq Y^2 kn/2\alpha$ for every vertex u.*

Theorem 4.8. *For any $\alpha > n$, the price of anarchy is $O(\sqrt{n/\alpha}\lg^{1+\lg 6} n)$ and the diameter of any equilibrium graph is $O(\lg^{\lg 6} n \cdot \sqrt{\alpha/n})$.*

By Theorem 4.8, we conclude the following:

Corollary 4.9. *For $\alpha = \Omega(n\lg^{2+2\lg 6} n) \approx \Omega(n\lg^{7.16} n)$, the price of anarchy is $O(1)$.*

5. Cooperative Version in General Graphs

In this section, we study the price of anarchy when only some links can be created, e.g., because of physical limitations. In this case, the social optimum is no longer simply a clique or a star.

We start by bounding the growth of distances from the host graph G to an arbitrary equilibrium graph G_s:

Lemma 5.1. *For any two vertices u and v in any equilibrium graph G_s, $d_{G_s}(u,v) = O(d_G(u,v) + \alpha^{1/3} d_G(u,v)^{2/3})$.*

Proof. Let $u = v_0, v_1, \ldots, v_k = v$ be a shortest path in G between u and v, so $k = d_G(u,v)$. Suppose that the distance between v_0 and v_i in G_s is d_i, for $0 \le i \le k$. We first prove that $d_{i+1} \le d_i + 1 + \sqrt{9\alpha/d_i}$ for $0 \le i < k$. If edge $\{v_i, v_{i+1}\}$ already exists in G_s, the inequality clearly holds. Otherwise, adding this edge decreases the distance between x and y by at least $\frac{d_{i+1}-d_i}{3}$, where x is a vertex whose distance is at most $\frac{d_{i+1}-d_i}{3} - 1$ from v_{i+1} and y is a vertex in a shortest path from v_i to v_0. Therefore any vertex x whose distance is at most $\frac{d_{i+1}-d_i}{3} - 1$ from v_{i+1} can pay $\frac{d_{i+1}-d_i}{3} d_i$ for this edge. Because this edge does not exist in G_s and because there are at least $\frac{d_{i+1}-d_i}{3}$ vertices of distance at most $\frac{d_{i+1}-d_i}{3} - 1$ from v_{i+1}, we conclude that $\left(\frac{d_{i+1}-d_i}{3}\right)^2 d_i \le \alpha$. Thus we have $d_{i+1} \le d_i + 1 + \sqrt{9\alpha/d_i}$ for $0 \le i < k$. Next we prove that $d_{i+1} \le d_i + 1 + 5\alpha^{1/3}$. If edge $\{v_i, v_{i+1}\}$ already exists in G_s, the inequality clearly holds. Otherwise, adding this edge decreases the distance between z and w by at least $\frac{d_{i+1}-d_i}{5}$, where z and w are two vertices whose distances from v_{i+1} and v_i, respectively, are less than $\frac{d_{i+1}-d_i}{5}$. There are at least at least $\left(\frac{d_{i+1}-d_i}{5}\right)^2$ pair of vertices like (z,w). Because the edge $\{v_i, v_{i+1}\}$ does not exist in G_s, we conclude that $\left(\frac{d_{i+1}-d_i}{5}\right)^3 \le \alpha$. Therefore $d_{i+1} \le d_i + 1 + 5\alpha^{1/3}$. Combining these two inequalities, we obtain $d_{i+1} \le d_i + 1 + \min\{\sqrt{9\alpha/d_i}, 5\alpha^{1/3}\}$.

Inductively we prove that $d_j \le 3j + 7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3}$. For $j \le 2$, the inequality is clear. Now suppose by induction that $d_j \le 3j + 7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3}$. If $d_j \le 2\alpha^{1/3}$, we reach the desired inequality using the inequality $d_{j+1} \le d_j + 1 + 5\alpha^{1/3}$. Otherwise, we know that $d_{j+1} \le d_j + 1 + \sqrt{9\alpha/d_j} = f(d_j)$ and to find the maximum of the function $f(d_j)$ over the domain $d_j \in [2\alpha^{1/3}, j + 7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3}]$, we should check f's critical points, including the endpoints of the domain interval and where f's derivative is zero. We reach three values for d_j: $2\alpha^{1/3}$, $j + 7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3}$, and $\left(\frac{9\alpha}{4}\right)^{1/3}$. Because the third value is not in the domain, we just need to check the first two values. The first value is also checked, so just the second value remains. For the second value, we have $d_{j+1} \le d_j + 1 + \sqrt{9\alpha/d_j} \le j + 7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3} + 1 + \sqrt{\frac{9\alpha}{j + 7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3}}} \le j + 1 + 7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3} + \sqrt{\frac{10\alpha}{5\alpha^{1/3} j^{2/3}}} \le j + 1 +$

$7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3} + \frac{\alpha^{1/3}\sqrt{2}}{j^{1/3}}$. Because $(j+1)^{2/3} - j^{2/3} = \frac{(j+1)^2 - j^2}{(j+1)^{4/3} + (j+1)^{2/3} j^{2/3} + j^{4/3}} \geq \frac{2j}{3(j+1)^{4/3}}$, we have $j+1+7\alpha^{1/3} + 5\alpha^{1/3} j^{2/3} + \frac{\alpha^{1/3}\sqrt{2}}{j^{1/3}} \leq j+1+7\alpha^{1/3} + 5\alpha^{1/3}(j+1)^{2/3} - 5\alpha^{1/3}\frac{2j}{3(j+1)^{4/3}} + \frac{\alpha^{1/3}\sqrt{2}}{j^{1/3}} \leq j+1+7\alpha^{1/3} + 5\alpha^{1/3}(j+1)^{2/3} - \frac{10\alpha^{1/3} j}{3j^{4/3}} + \frac{\alpha^{1/3}\sqrt{2}}{j^{1/3}} \leq j+1+7\alpha^{1/3} + 5\alpha^{1/3}(j+1)^{2/3}$.

Note that $j+1 > 2$ and $d_k = d_{G_s}(u,v)$. Therefore $d_{G_s}(u,v)$ is at most $O(d_G(u,v) + \alpha^{1/3} d_G(u,v)^{2/3})$ and the desired inequality is proved. □

Using this Lemma 5.1, we prove two different bounds relating the sum of all pairwise distances in the two graphs:

Corollary 5.2. *For any equilibrium graph G_s, $\sum_{u,v \in V(G)} d_{G_s}(u,v) = O(\alpha^{1/3}) \cdot \sum_{u,v \in V(G)} d_G(u,v)$.*

Theorem 5.3. *For any equilibrium graph G_s, $\sum_{u,v \in V(G)} d_{G_s}(u,v) \leq \min\{O(n^{1/3})(\alpha n + \sum_{u,v \in V(G)} d_G(u,v)), n^3\}$.*

Now we can bound the price of anarchy for the various ranges of α, combining Corollary 5.2, Theorem 5.3, and Lemma 3.4, with different choices of g.

Theorem 5.4. *In the cooperative network creation game in general graphs, the price of anarchy is at most*

(a) $O(\alpha^{1/3})$ *for $\alpha < n$ [$g = 6$ in Lemma 3.4 and Corollary 5.2],*
(b) $O(n^{1/3})$ *for $n \leq \alpha \leq n^{5/3}$ [$g = 6$ in Lemma 3.4 and Theorem 5.3],*
(c) $O(\frac{n^2}{\alpha})$ *for $n^{5/3} \leq \alpha < n^{2-\varepsilon}$ [$g = 2/\varepsilon$ in Lemma 3.4 and Theorem 5.3], and*
(d) $O(\frac{n^2}{\alpha} \lg n)$ *for $n^2 \leq \alpha$ [$g = \lg n$ in Lemma 3.4 and Theorem 5.3].*

6. Unilateral Version in General Graphs

Next we consider how a general host graph affects the unilateral version of the problem. Some proofs are similar to proofs for the cooperative version in Section 5 and hence omitted.

Lemma 6.1. *For any two vertices u and v in any equilibrium graph G_s, $d_{G_s}(u,v) = O(d_G(u,v) + \alpha^{1/2} d_G(u,v)^{1/2})$.*

Again we relate the sum of all pairwise distances in the two graphs:

Corollary 6.2. *For any equilibrium graph G_s, $\sum_{u,v \in V(G)} d_{G_s}(u,v) = O(\alpha^{1/2}) \cdot \sum_{u,v \in V(G)} D_G(u,v)$.*

Theorem 6.3. *For any equilibrium graph G_s, $\sum_{u,v \in V(G_s)} d_{G_s}(u,v) \leq \min\{O(n^{1/2})(\alpha n + \sum_{u,v \in V(G)} D_G(u,v)), n^3\}$.*

To conclude bounds on the price of anarchy, we now use Lemma 3.2 in place of Lemma 3.4, combined with Corollary 6.2 and Theorem 6.3.

Theorem 6.4. *For $\alpha \geq n$, the price of anarchy is at most* $\min\{O(n^{1/2}), \frac{n^2}{\alpha}\}$.

Theorem 6.5. *For $\alpha < n$, the price of anarchy is at most $O(\alpha^{1/2})$.*

7. Lower Bounds in General Graphs

In this section, we prove polynomial lower bounds on the price of anarchy for general host graphs, first for the cooperative version and second for the unilateral version.

Theorem 7.1. *The price of anarchy in the cooperative game is* $\Omega(\min\{\sqrt{\frac{\alpha}{n}}, \frac{n^2}{\alpha}\})$.

Proof. For $\alpha = O(n)$ or $\alpha = \Omega(n^2)$, the claim is clear. Otherwise, let $k = \sqrt{\frac{\alpha}{12n}} \geq 2$. Thus $k = O(\sqrt{n})$. We construct graph $G_{k,l}$ as follows; see Figure 2. Start with $2l$ vertices $v_1, v_2, \ldots, v_{2l}$ connected in a cycle. For any $1 \leq i \leq 2l$, insert a path P_i of k edges between v_i and v_{i+1} (where we define $v_{2l+1} = v_1$). For any $1 \leq i \leq l$, insert a path Q_i of k edges between v_{2i} and v_{2i+2} (where we define $v_{2l+2} = v_2$). Therefore there are $n = (3k-1)l$ vertices and $(3k+2)l$ edges in $G_{k,l}$, so $l = n/(3k-1)$.

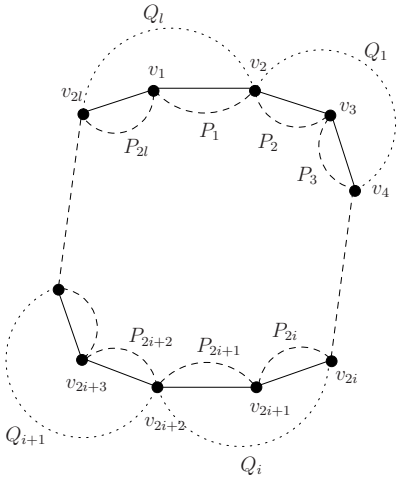


Figure 2: Lower bound graph.

For simplicity, let G denote $G_{k,l}$ in the rest of the proof. Let G_1 be a spanning connected subgraph of G that contains exactly one cycle, namely, $(v_1, v_2, \ldots, v_{2l}, v_1)$; in other words, we remove from G exactly one edge from each path P_i and Q_i. Let G_2 be a spanning connected subgraph of G that contains exactly one cycle, formed by the concatenation of $Q_1, Q_2, \ldots, Q_l$, and contains none of the edges $\{v_i, v_{i+1}\}$, for $1 \leq i \leq 2l$; for example, we remove from G exactly one edge from every P_{2i} and every edge $\{v_i, v_{i+1}\}$.

Next we prove that G_2 is an equilibrium. For any $1 \leq i \leq l$, removing any edge of path Q_i increases the distance between its endpoints and at least $n/6$ vertices by at least $\frac{lk}{3} \geq n/6$. Because $\alpha = o(n^2)$, we have $\alpha < \frac{n}{6}\frac{n}{6}$, so if we assign this edge to be bought solely by one of its endpoints, then this owner will not delete the edge. Removing other edges makes G_2 disconnected. For any $1 \leq i \leq l$, adding an edge of path P_{2i} or path P_{2i+1} or edge $\{v_{2i}, v_{2i+1}\}$ or edge $\{v_{2i+1}, v_{2i+2}\}$ to G_2 decreases only the distances from some vertices of paths P_{2i} or P_{2i+1} to the other vertices. There are at most $n(2k-1)$ such pairs. Adding such an edge can decrease each of these distance by at most $3k-1$. But we know that $\alpha \geq 12nk^2 > 2n(2k-1)(3k-1)$, so the price of the edge is more than its total benefit among all nodes, and thus the edge will not be created by any coalition.

The cost of G_1 is equal to $O(\alpha n + n^2(k+l)) = O(\alpha n + n^2(k+\frac{n}{k}))$ and the cost of G_2 is $\Omega(\alpha n + n^2(k+lk)) = \Omega(\alpha n + n^3)$. The cost of the social optimum is at most the cost of G_1, so the price of anarchy is at least $\Omega(\frac{n^3}{\alpha n + n^3/k + kn^2}) = \Omega(\min\{\frac{n^2}{\alpha}, k, \frac{n}{k}\})$. Because $k = O(\sqrt{n})$, the price of anarchy is at least $\Omega(\min\{\frac{n^2}{\alpha}, k\}) = \Omega(\min\{\frac{n^2}{\alpha}, \sqrt{\frac{\alpha}{n}}\})$. □

Theorem 7.2. *The price of anarchy in unilateral games is* $\Omega(\min\{\frac{\alpha}{n}, \frac{n^2}{\alpha}\})$.

The proof uses a construction similar to Theorem 7.1.

References

[1] S. Albers. On the value of coordination in network design. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 294–303, 2008.

[2] S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty. On Nash equilibria for a network creation game. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 89–98, 2006.
[3] N. Andelman, M. Feldman, and Y. Mansour. Strong price of anarchy. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 189–198, 2007.
[4] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 295–304, 2004.
[5] E. Anshelevich, A. Dasgupta, E. Tardos, and T. Wexler. Near-optimal network design with selfish agents. In *Proc. 35th Annual ACM Symposium on Theory of Computing*, pages 511–520, 2003.
[6] R. J. Aumann. Acceptable points in general cooperative n-person games. In *Contributions to the Theory of Games*, volume 4. Princeton University Press, 1959.
[7] Y. Bartal. On approximating arbitrary metrices by tree metrics. In *Proc. 13th Annual ACM Symposium on Theory of Computing*, pages 161–168, 1998.
[8] B.-G. Chun, R. Fonseca, I. Stoica, and J. Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In *Proc. 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, 2004.
[9] J. Corbo and D. Parkes. The price of selfish behavior in bilateral network formation. In *Proc. 24th Annual ACM Symposium on Principles of Distributed Computing*, pages 99–107, 2005.
[10] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. In *Proc. 13th Annual ACM-SIAM symposium on Discrete Algorithms*, pages 413–420, 2002.
[11] E. D. Demaine, M. Hajiaghayi, H. Mahini, and M. Zadimoghaddam. The price of anarchy in network creation games. In *Proc. 26th Annual ACM Symposium on Principles of Distributed Computing*, pages 292–298, 2007.
[12] R. D. Dutton and R. C. Brigham. Edges in graphs with large girth. *Graphs and Combinatorics*, 7(4):315–321, 1991.
[13] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. In *Proc. 37th Annual ACM Symposium on Theory of Computing*, pages 494–503, 2005.
[14] E. Even-Dar and M. Kearns. A small world threshold for economic network formation. In *Proc. 20th Annual Conference on Neural Information Processing Systems*, pages 385–392, 2006.
[15] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *Proc. 22nd Annual Symposium on Principles of Distributed Computing*, pages 347–351, 2003.
[16] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.
[17] Y. Halevi and Y. Mansour. A network creation game with nonuniform interests. In *Proc. 3rd International Workshop on Internet and Network Economics*, LNCS 4858, pages 287–292, 2007.
[18] M. O. Jackson. A survey of models of network formation: Stability and efficiency. In G. Demange and M. Wooders, editors, *Group Formation in Economics; Networks, Clubs and Coalitions*. Cambridge University Press, 2003.
[19] J. M. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.
[20] E. Koutsoupias and C. H. Papadimitriou. Worst-case equilibria. In *Proc. 16th Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 1563, pages 404–413, 1999.
[21] N. Laoutaris, L. J. Poplawski, R. Rajaraman, R. Sundaram, and S.-H. Teng. Bounded budget connection (BBC) games or how to make friends and influence people, on a budget. In *Proc. 27th ACM Symposium on Principles of Distributed Computing*, pages 165–174, 2008.
[22] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.
[23] H. Lin. On the price of anarchy of a network creation game. Class final project, December 2003.
[24] J. Nash. Equilibrium points in n-person games. *Proc. National Academy of Sciences*, 36(1):48–49, 1950.
[25] J. Nash. Non-cooperative games. *Annals of Mathematics (2)*, 54:286–295, 1951.
[26] C. Papadimitriou. Algorithms, games, and the internet. In *Proc. 33rd Annual ACM Symposium on Theory of Computing*, pages 749–753, 2001.
[27] T. Roughgarden. The price of anarchy is independent of the network topology. In *Proc. 34th Annual ACM Symposium on Theory of Computing*, pages 428–437, 2002.
[28] T. Roughgarden. *Selfish Routing*. PhD thesis, Cornell University, 2002. Published as *Selfish Routing and the Price of Anarchy*, MIT Press, 2005.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 313–324
www.stacs-conf.org

ERROR-CORRECTING DATA STRUCTURES

RONALD DE WOLF [1]

[1] CWI. Kruislaan 413. 1098SJ Amsterdam. The Netherlands. E-mail address: rdewolf@cwi.nl

Abstract. We study data structures in the presence of adversarial noise. We want to encode a given object in a succinct data structure that enables us to efficiently answer specific queries about the object, even if the data structure has been corrupted by a constant fraction of errors. This new model is the common generalization of (static) data structures and locally decodable error-correcting codes. The main issue is the tradeoff between the space used by the data structure and the time (number of probes) needed to answer a query about the encoded object. We prove a number of upper and lower bounds on various natural error-correcting data structure problems. In particular, we show that the optimal length of error-correcting data structures for the Membership problem (where we want to store subsets of size s from a universe of size n) is closely related to the optimal length of locally decodable codes for s-bit strings.

1. Introduction

Data structures deal with one of the most fundamental questions of computer science: how can we store certain objects in a way that is both space-efficient and that enables us to efficiently answer questions about the object? Thus, for instance, it makes sense to store a set as an ordered list or as a heap-structure, because this is space-efficient and allows us to determine quickly (in time logarithmic in the size of the set) whether a certain element is in the set or not. ¿From a complexity-theoretic point of view, the aim is usually to study the tradeoff between the two main resources of the data structure: the length/size of the data structure (storage space) and the efficiency with which we can answer specific queries about the stored object. To make this precise, we measure the length of the data structure in bits, and measure the efficiency of query-answering in the number of *probes*, i.e., the number of bit-positions in the data structure that we look at in order to answer a query. The following is adapted from Miltersen's survey [Mil99]:

Definition 1.1. Let D be a set of data items, Q be a set of queries, A be a set of answers, and $f : D \times Q \to A$. A (p, ε)-*data structure* for f of length N is a map $\phi : D \to \{0,1\}^N$ for which there is a randomized algorithm $\mathcal{A}$ that makes at most p probes to its oracle and satisfies $\Pr[\mathcal{A}^{\phi(x)}(q) = f(x,q)] \geq 1 - \varepsilon$ for every $q \in Q$ and $x \in D$.

1998 ACM Subject Classification: E1, E4.

Key words and phrases: data structures, error-correcting codes, locally decodable codes, membership.

Partially supported by a Veni grant from the Netherlands Organization for Scientific Research (NWO), and by the European Commission under Integrated Project QAP, IST 015848.

Usually we will study the case $D \subseteq \{0,1\}^n$ and $A = \{0,1\}$. Most standard data structures taught in undergraduate computer science are deterministic, and hence have error probability $\varepsilon = 0$. As mentioned, the main complexity issue here is the tradeoff between N and p. Some data structure problems that we will consider are the following:

- EQUALITY. $D = Q = \{0,1\}^n$, and $f(x,y) = 1$ if $x = y$, $f(x,y) = 0$ if $x \neq y$. This is not a terribly interesting data structure problem in itself, since for every x there is only one query y for which the answer is '1'; we merely mention this data structure problem here because it will be used to illustrate some definitions later on.
- MEMBERSHIP. $D = \{x \in \{0,1\}^n : \text{Hamming weight } |x| \leq s\}$, $Q = [n] := \{1,\ldots,n\}$, and $f(x,i) = x_i$. In other words, x corresponds to a set of size at most s from a universe of size n, and we want to store the set in a way that easily allows us to make membership queries. This is probably the most basic and widely-studied data structure problem of them all [FKS84, Yao81, BMRV00, RSV02]. Note that for $s = 1$ this is EQUALITY on $\log n$ bits, while for $s = n$ it is the general MEMBERSHIP problem without constraints on the set.
- SUBSTRING. $D = \{0,1\}^n$, $Q = \{y \in \{0,1\}^n : |y| \leq r\}$, $f(x,y) = x_y$, where x_y is the $|y|$-bit substring of x indexed by the 1-bits of y (e.g., $1010_{0110} = 01$). For $r = 1$ it is MEMBERSHIP.
- INNER PRODUCT ($\mathrm{IP}_{n,r}$). $D = \{0,1\}^n$, $Q = \{y \in \{0,1\}^n : |y| \leq r\}$ and $f(x,y) = x \cdot y \bmod 2$. This problem is among the hardest Boolean problems where the answer depends on at most r bits of x (again, for $r = 1$ it is MEMBERSHIP).

More complicated data structure problems such as RANK, PREDECESSOR, NEAREST NEIGHBOR have also been studied a lot, but we will not consider them here.

One issue that the above definition ignores, is the issue of *noise*. Memory and storage devices are not perfect: the world is full of cosmic rays, small earthquakes, random (quantum) events, bypassing trams, etc., that can cause a few errors here and there. Another potential source of noise is transmission of the data structure over some noisy channel. Of course, better hardware can partly mitigate these effects, but in many situations it is realistic to expect a small fraction of the bits in the storage space to become corrupted over time. Our goal in this paper is to study *error-correcting* data structures. These still enable efficient computation of $f(x,q)$ from the stored data structure $\phi(x)$, even if the latter has been corrupted by a constant fraction of errors. In analogy with the usual setting for error-correcting codes [MS77, vL98], we will take a pessimistic, adversarial view of errors here: we want to be able to deal with a constant fraction of errors *no matter where they are placed*. Formally, we define error-correcting data structures as follows.

Definition 1.2. Let D be a set of data items, Q be a set of queries, A be a set of answers, and $f : D \times Q \to A$. A (p,δ,ε)-*error-correcting data structure* for f of length N is a map $\phi : D \to \{0,1\}^N$ for which there is a randomized algorithm $\mathcal{A}$ that makes at most p probes to its oracle and satisfies $\Pr[\mathcal{A}^y(q) = f(x,q)] \geq 1 - \varepsilon$ for every $q \in Q$, every $x \in D$, and every $y \in \{0,1\}^N$ at distance $\Delta(y,\phi(x)) \leq \delta N$.

Definition 1.1 is the special case of Definition 1.2 where $\delta = 0$.[1] Note that if $\delta > 0$ then the adversary can always set the errors in a way that gives the decoder $\mathcal{A}$ a non-zero error probability. Hence the setting with bounded error probability is the natural one for error-correcting data structures. This contrasts with the standard noiseless setting, where one usually considers deterministic structures.

A simple example of an efficient error-correcting data structure is for EQUALITY: encode x with a good error-correcting code $\phi(x)$. Then $N = O(n)$, and we can decode by one probe: given y, probe $\phi(x)_j$ for uniformly chosen $j \in [N]$, compare it with $\phi(y)_j$, and output 1 iff these two bits are equal. If up to a δ-fraction of the bits in $\phi(x)$ are corrupted, then we will give the correct answer with probability $1-\delta$ in the case $x = y$. If the distance between any two codewords is close to $N/2$ (which is true for instance for a random linear code), then we will give the correct answer with probability about $1/2-\delta$ in the case $x \neq y$. These two probabilities can be balanced to 2-sided error $\varepsilon = 1/3 + 2\delta/3$. The error can be reduced further by allowing more than one probe.

We only deal with so-called *static* data structures here: we do not worry about updating the x that we are encoding. What about *dynamic* data structures, which allow efficient updates as well as efficient queries to the encoded object? Note that if data-items x and x' are distinguishable in the sense that $f(x,q) \neq f(x',q)$ for at least one query $q \in Q$, then their respective error-correcting encodings $\phi(x)$ and $\phi(x')$ will have distance $\Omega(N)$.[2] Hence updating the encoded data from x to x' will require $\Omega(N)$ changes in the data structure, which shows that a dynamical version of our model of error-correcting data structures with efficient updates is not possible.

Error-correcting data structures not only generalize the standard (static) data structures (Definition 1.1), but they also generalize *locally decodable codes*, defined as:

Definition 1.3. A (p,δ,ε)-*locally decodable code* (LDC) of length N is a map $\phi : \{0,1\}^n \to \{0,1\}^N$ for which there is a randomized algorithm $\mathcal{A}$ that makes at most p probes to its oracle and satisfies $\Pr[\mathcal{A}^y(i) = x_i] \geq 1-\varepsilon$ for every $i \in [n]$, every $x \in \{0,1\}^n$, and every $y \in \{0,1\}^N$ at distance $\Delta(y,\phi(x)) \leq \delta N$.

Note that a (p,δ,ε)-error-correcting data structure for MEMBERSHIP (with $s = n$) is exactly a (p,δ,ε)-locally decodable code. Much work has been done on LDCs, but their length-vs-probes tradeoff is still largely unknown for $p \geq 3$. We refer to [Tre04] and the references therein.

LDCs address only a very simple type of data structure problem: we have an n-bit "database" and want to be able to retrieve individual bits from it. In practice, databases have more structure and complexity, and one usually asks more complicated queries, such as retrieving all records within a certain range. Our more general notion of error-correcting data structures enables a study of such more practical data structure problems in the presence of adversarial noise.

[1] As [BMRV00, end of Section 1.1] notes, a data structure can be viewed as locally decodable source code. With this information-theoretic point of view, an *error-correcting* data structure is a locally decodable combined source-*channel* code, and our results for MEMBERSHIP show that one can sometimes do better than combining the best source code with the best channel code. We thank one of the anonymous referees for pointing this out.

[2] Hence if all pairs $x, x' \in D$ are distinguishable (which is usually the case), ϕ is an error-correcting code.

Comment on terminology. The terminologies used in the data-structure and LDC-literature conflict at various points, and we needed to reconcile them somehow. To avoid confusion, let us repeat here the choices we made. We reserve the term "query" for the question q one asks about the encoded data x, while accesses to bits of the data structure are called "probes" (in contrast, these are usually called "queries" in the LDC-literature). The number of probes is denoted by p. We use n for the number of bits of the data item x (in contrast with the literature about MEMBERSHIP, which mostly uses m for the size of the universe and n for the size of the set). We use N for the length of the data structure (while the LDC-literature mostly uses m, except for Yekhanin [Yek07] who uses N as we do). We use the term "decoder" for the algorithm $\mathcal{A}$. Another issue is that ε is sometimes used as the error probability (in which case one wants $\varepsilon \approx 0$), and sometimes as the bias away from 1/2 (in which case one wants $\varepsilon \approx 1/2$). We use the former.

1.1. Our results

If one subscribes to the approach to errors taken in the area of error-correcting codes, then our definition of error-correcting data structures seems a very natural one. Yet, to our knowledge, it is new and has not been studied before (see Section 1.2 for other approaches).

1.1.1. MEMBERSHIP. The most basic data structure problem is probably MEMBERSHIP. Fortunately, our main positive result for error-correcting data structures applies to this.

Fix some number of probes p, noise level δ, and allowed error probability ε, and consider the minimal length of p-probe error-correcting data structures for s-out-of-n MEMBERSHIP. Let us call this minimal length $\mathrm{MEM}(p, s, n)$. A first observation is that such a data structure is actually a locally decodable code for s bits: just restrict attention to n-bit strings whose last $n - s$ bits are all 0. Hence, with $\mathrm{LDC}(p, s)$ denoting the minimal length among all p-probe LDCs that encode s bits (for our fixed ε, δ), we immediately get the obvious lower bound

$$\mathrm{LDC}(p, s) \leq \mathrm{MEM}(p, s, n).$$

This bound is close to optimal if $s \approx n$. Another trivial lower bound comes from the observation that our data structure for MEMBERSHIP is a map with domain of size $B(n, s) := \sum_{i=0}^{s} \binom{n}{i}$ and range of size 2^N that has to be injective. Hence we get another obvious bound

$$\Omega(s \log(n/s)) \leq \log B(n, s) \leq \mathrm{MEM}(p, s, n).$$

What about *upper* bounds? Something that one can always do to construct error-correcting data structures for any problem, is to take the optimal non-error-correcting p_1-probe construction and encode it with a p_2-probe LDC. If the error probability of the LDC is much smaller than $1/p_1$, then we can just run the decoder for the non-error-correcting structure, replacing each of its p_1 probes by p_2 probes to the LDC. This gives an error-correcting data structure with $p = p_1 p_2$ probes. In the case of MEMBERSHIP, the optimal non-error-correcting data structure of Buhrman et al. [BMRV00] uses only 1 probe and $O(s \log n)$ bits. Encoding this with the best possible p-probe LDC gives error-correcting data structures for MEMBERSHIP of length $\mathrm{LDC}(p, O(s \log n))$. For instance for $p = 2$ we can use the Hadamard code[3] for s bits, giving upper bound $\mathrm{MEM}(2, s, n) \leq \exp(O(s \log n))$.

[3]The Hadamard code of $x \in \{0,1\}^s$ is the codeword of length 2^s obtained by concatenating the bits $x \cdot y \pmod 2$ for all $y \in \{0,1\}^s$. It can be decoded by two probes, since for every $y \in \{0,1\}^s$ we have $(x \cdot y) \oplus (x \cdot (y \oplus e_i)) = x_i$. Picking y at random, decoding from a δ-corrupted codeword will be correct with

Our main positive result in Section 2 says that something much better is possible—the max of the above two lower bounds is not far from optimal. Slightly simplifying[4], we prove

$$\mathrm{MEM}(p, s, n) \leq O(\mathrm{LDC}(p, 1000s) \log n).$$

In other words, if we have a decent p-probe LDC for encoding $O(s)$-bit strings, then we can use this to encode sets of size s from a much larger universe $[n]$, at the expense of blowing up our data structure by only a factor of $\log n$. For instance, for $p = 2$ probes we get $\mathrm{MEM}(2, s, n) \leq \exp(O(s)) \log n$ from the Hadamard code, which is much better than the earlier $\exp(O(s \log n))$. For $p = 3$ probes, we get $\mathrm{MEM}(3, s, n) \leq \exp(\exp(\sqrt{\log s})) \log n$ from Efremenko's recent 3-probe LDC [Efr08] (which improved Yekhanin's breakthrough construction [Yek07]). Our construction relies heavily on the MEMBERSHIP construction of [BMRV00]. Note that the near-tightness of the above upper and lower bounds implies that progress (meaning better upper and/or lower bounds) on locally decodable codes for any number of probes is *equivalent* to progress on error-correcting data structures for s-out-of-n MEMBERSHIP.

1.1.2. INNER PRODUCT. In Section 3 we analyze the inner product problem, where we are encoding $x \in \{0,1\}^n$ and want to be able to compute the dot product $x \cdot y \pmod 2$, for any $y \in \{0,1\}^n$ of weight at most r. We first study the non-error-correcting setting, where we can prove nearly matching upper and lower bounds (this is not the error-correcting setting, but provides something to compare it with). Clearly, a trivial 1-probe data structure is to store the answers to all $B(n, r)$ possible queries separately. In Section 3.1 we use a discrepancy argument from communication complexity to prove a lower bound of about $B(n, r)^{1/p}$ on the length of p-probe data structures. This shows that the trivial solution is essentially optimal if $p = 1$. We also construct various p-probe error-correcting data structures for inner product. For small p and large r, their length is not much worse than the best non-error-correcting structures. The upshot is that inner product is a problem where data structures can sometimes be made error-correcting at little extra cost compared to the non-error-correcting case—admittedly, this is mostly because the non-error-correcting solutions for $\mathrm{IP}_{n,r}$ are already very expensive in terms of length.

1.2. Related work

Much work has of course been done on locally decodable codes, a.k.a. error-correcting data structures for the MEMBERSHIP problem without constraints on the set size [Tre04]. However, the error-correcting version of s-out-of-n MEMBERSHIP ("storing sparse tables") or of other possible data structure problems has not been studied before.[5] Here we briefly describe some other approaches to data structures in the presence of memory errors. There is also much work on data structures with faulty *processors*, but we won't discuss that.

probability at least $1 - 2\delta$, because both probes y and $y \oplus e_i$ are individually random and hence probe a corrupted entry with probability at most δ. This exponential length is optimal for 2-probe LDCs [KW04].

[4]Our actual result, Theorem 2.2, is a bit dirtier, with some deterioration in the error and noise parameters.

[5]Using the connection between information-theoretical private information retrieval and locally decodable codes, one may derive some error-correcting data structures from the PIR results of [CIK+01]. However, the resulting structures seem fairly weak.

Fault-tolerant pointer-based data structures. Aumann and Bender [AB96] study fault-tolerant versions of *pointer-based* data structures. They define a pointer-based data structure as a directed graph where the edges are pointers, and the nodes come in two types: information nodes carry real data, while auxiliary nodes carry auxiliary or structural data. An *error* is the destruction of a node and its outgoing edges. They assume such an error is detected when accessing the node. Even a few errors may be very harmful to pointer-based data structures: for instance, losing one pointer halfway a standard linked list means we lose the second half of the list. They call a data structure (d, g)-*fault-tolerant* (where d is an integer that upper bounds the number of errors, and g is a function) if $f \leq d$ errors cause at most $g(f)$ information nodes to be lost.

Aumann and Bender present fault-tolerant *stacks* with $g(f) = O(f)$, and fault-tolerant *linked lists* and *binary search trees* with $g(f) = O(f \log d)$, with only a constant-factor overhead in the size of the data structure, and small computational overhead. Note, however, that their error-correcting demands are weaker than ours: we require that *no* part of the data is lost (every query should be answered with high success probability), even in the presence of a constant fraction of errors. Of course, we pay for that in terms of length.

Faulty-memory RAM model. An alternative model of error-correcting data structures is the "faulty-memory RAM model", introduced by Finocchi and Italiano [FI04]. In this model, one assumes there are $O(1)$ incorruptible memory cells available. This is justified by the fact that CPU registers are much more robust than other kinds of memory. On the other hand, all other memory cells can be faulty—including the ones used by the algorithm that is answering queries (something our model does not consider). The model assumes an upper bound Δ on the number of errors.

Finocchi, Grandoni, and Italiano described essentially optimal resilient algorithms for *sorting* that work in $O(n \log n + \Delta^2)$ time with Δ up to about $\sqrt{n}$; and for *searching* in $\Theta(\log n + \Delta)$ time. There is a lot of recent work in this model: Jørgenson et al. [JMM07] study resilient *priority queues*, Finocchi et al. [FGI07] study resilient *search trees*, and Brodal et al. [BFF+07] study resilient *dictionaries*. This interesting model allows for more efficient data structures than our model, but its disadvantages are also clear: it assumes a small number of incorruptible cells, which may not be available in many practical situations (for instance when the whole data structure is stored on a hard disk), and the constructions mentioned above cannot deal well with a constant noise rate.

2. The Membership problem

2.1. Noiseless case: the BMRV data structure for Membership

Our error-correcting data structures for Membership rely heavily on the construction of Buhrman et al. [BMRV00], whose relevant properties we sketch here. Their structure is obtained using the probabilistic method. Explicit but slightly less efficient structures were subsequently given by Ta-Shma [TS02].

The BMRV-structure maps $x \in \{0,1\}^n$ (of weight $\leq s$) to a string $y := y(x) \in \{0,1\}^{n'}$ of length $n' = \frac{100}{\varepsilon^2} s \log n$ that can be decoded with one probe if $\delta = 0$. More precisely, for every $i \in [n]$ there is a set $P_i \subseteq [n']$ of size $\log(n)/\varepsilon$, such that for every x of weight $\leq s$:

$$\Pr_{j \in P_i}[y_j = x_i] \geq 1 - \varepsilon, \tag{2.1}$$

where the probability is over a uniform index $j \in P_i$. For fixed ε, the length $n' = O(s \log n)$ is optimal up to a constant factor, because clearly $\log \binom{n}{s}$ is a lower bound.

2.2. Noisy case: 1 probe

For the noiseless case, the BMRV data structure has information-theoretically optimal length $O(s \log n)$ and decodes with the minimal number of probes (one). This can also be achieved in the error-correcting case if $s = 1$: then we just have the EQUALITY problem, for which see the remark following Definition 1.2. For larger s, one can observe that the BMRV-structure still works with high probability if $\delta \ll 1/s$: in that case the total number of errors is $\delta n' \ll \log n$, so for each i, most bits in the $\Theta(\log n)$-set P_i are uncorrupted.

Theorem 2.1 (BMRV). *There exist* $(1, \Omega(1/s), 1/4)$*-error-correcting data structures for* MEMBERSHIP *of length* $N = O(s \log n)$.

This only works if $\delta \ll 1/s$, which is actually close to optimal, as follows. An s-bit LDC can be embedded in an error-correcting data structure for MEMBERSHIP, hence it follows from Katz-Trevisan's [KT00, Theorem 3] that there are no 1-probe error-correcting data structures for MEMBERSHIP if $s > 1/(\delta(1 - H(\varepsilon)))$ (where $H(\cdot)$ denotes binary entropy). In sum, there are 1-probe error-correcting data structures for MEMBERSHIP of information-theoretically optimal length if $\delta \ll 1/s$. In contrast, if $\delta \gg 1/s$ then there are no 1-probe error-correcting data structures at all, not even of exponential length.

2.3. Noisy case: $p > 1$ probes

As we argued in the introduction, for fixed ε and δ there is an easy lower bound on the length N of p-probe error-correcting data structures for s-out-of-n MEMBERSHIP:

$$N \geq \max\left(\mathrm{LDC}(p, s), \log \sum_{i=0}^{s} \binom{n}{i}\right).$$

Our nearly matching upper bound, below, uses the ε-error data structure of [BMRV00] for some small fixed ε. A simple way to obtain a p-probe error-correcting data structure is just to encode their $O(s \log n)$-bit string y with the optimal p-probe LDC (with error ε', say), which gives length $\mathrm{LDC}(p, O(s \log n))$. The one probe to y is replaced by p probes to the LDC. By the union bound, the error probability of the overall construction is at most $\varepsilon + \varepsilon'$. This, however, achieves more than we need: this structure enables us to recover y_j for every j, whereas it would suffice to recover y_j for most $j \in P_i$ (for each $i \in [n]$).

Definition of the data structure and decoder. To construct a shorter error-correcting data structure, we proceed as follows. Let δ be a small constant (e.g. 1/10000); this is the noise level we want our final data structure for MEMBERSHIP to protect against. Consider the BMRV-structure for s-out-of-n MEMBERSHIP, with error probability at most 1/10. Then $n' = 10000 s \log n$ is its length, and $b = 10 \log n$ is the size of each of the sets P_i. Apply now a random permutation π to y (we show below that π can be fixed to a specific permutation). View the resulting n'-bit string as made up of $b = 10 \log n$ consecutive blocks of $1000s$ bits each. We encode each block with the optimal $(p, 100\delta, 1/100)$-LDC that encodes $1000s$ bits. Let ℓ be the length of this LDC. This gives overall length $N = 10\ell \log n$. The decoding procedure is as follows. Randomly choose a $k \in [b]$. This picks out one of the blocks. If this kth block contains exactly one $j \in P_i$ then recover y_j from the (possibly corrupted) LDC

for that block, using the p-probe LDC-decoder, and output y_j. If the kth block contains 0 or more than 1 elements from P_i, then output a uniformly random bit.

Analysis. Our goal below is to show that we can fix the permutation π such that for at least $n/20$ of the indices $i \in [n]$, this procedure has good probability of correctly decoding x_i (for all x of weight $\leq s$). The intuition is as follows. Thanks to the random permutation and the fact that $|P_i|$ equals the number of blocks, the expected intersection between P_i and a block is exactly 1. Hence for many $i \in [n]$, many blocks will contain exactly one index $j \in P_i$. Moreover, for most blocks, their LDC-encoding won't have too many errors, hence we can recover y_j using the LDC-decoder for that block. Since $y_j = x_i$ for 90% of the $j \in P_i$, we usually recover x_i.

To make this precise, call $k \in [b]$ "good for i" if block k contains *exactly one* $j \in P_i$, and let X_{ik} be the indicator random variable for this event. Call $i \in [n]$ "good" if at least $b/4$ of the blocks are good for i (i.e., $\sum_{k\in[b]} X_{ik} \geq b/4$), and let X_i be the indicator random variable for this event. The expected value (over uniformly random π) of each X_{ik} is the probability that if we randomly place b balls into ab positions (a is the block-size $1000s$), then there is exactly one ball among the a positions of the first block, and the other $b-1$ balls are in the last $ab-a$ positions. This is

$$\frac{a\binom{ab-a}{b-1}}{\binom{ab}{b}} = \frac{(ab-b)(ab-b-1)\cdots(ab-b-a+2)}{(ab-1)(ab-2)\cdots(ab-a+1)} \geq \left(\frac{ab-b-a+2}{ab-a+1}\right)^{a-1} \geq \left(1-\frac{1}{a-1}\right)^{a-1}$$

The righthand side goes to $1/e \approx 0.37$ with large a, so we can safely lower bound it by $3/10$. Then, using linearity of expectation:

$$\frac{3bn}{10} \leq \mathrm{Exp}\left[\sum_{i\in[n],k\in[b]} X_{ik}\right] \leq b \cdot \mathrm{Exp}\left[\sum_{i\in[n]} X_i\right] + \frac{b}{4}\left(n - \mathrm{Exp}\left[\sum_{i\in[n]} X_i\right]\right),$$

which implies $\mathrm{Exp}\left[\sum_{i\in[n]} X_i\right] \geq \frac{n}{20}$. Hence we can fix one permutation π such that at least $n/20$ of the indices i are good.

For every index i, at least 90% of all $j \in P_i$ satisfy $y_j = x_i$. Hence for a good index i, with probability at least $1/4 - 1/10$ we pick a k such that the kth block is good for i *and* the unique $j \in P_i$ in the kth block satisfies $y_j = x_i$. By Markov's inequality, the probability that the picked block has more than a 100δ-fraction of errors, is less than $1/100$. If the fraction of errors is at most 100δ, then our LDC-decoder recovers the relevant bit y_j with probability $99/100$. Hence the overall probability of outputting the correct x_i is at least

$$\frac{3}{4}\cdot\frac{1}{2} + \left(\frac{1}{4} - \frac{1}{10} - \frac{1}{100}\right)\cdot\frac{99}{100} > \frac{51}{100}.$$

We end up with an error-correcting data structure for Membership for a universe of size $n/20$ instead of n elements, but we can fix this by starting with the BMRV-structure for $20n$ bits.

We summarize this construction in a theorem:

Theorem 2.2. *If there exists a $(p, 100\delta, 1/100)$-LDC of length ℓ that encodes $1000s$ bits, then there exists a $(p, \delta, 49/100)$-error-correcting data structure of length $O(\ell \log n)$ for the s-out-of-n* Membership *problem.*

The error and noise parameters of this new structure are not great, but they can be improved by more careful analysis. We here sketch a better solution without giving all technical details. Suppose we change the decoding procedure for x_i as follows: pick $j \in P_i$ uniformly at random, decode y_j from the LDC of the block where y_j sits, and output the result. There are three sources of error here: (1) the BMRV-structure makes a mistake (i.e., j happens to be such that $y_j \neq x_i$), (2) the LDC-decoder fails because there is too much noise on the LDC that we are decoding from, (3) the LDC-decoder fails even though there is not too much noise on it. The 2nd kind is hardest to analyze. The adversary will do best if he puts just a bit more than the tolerable noise-level on the encodings of blocks that contain the most $j \in P_i$, thereby "destroying" those encodings.

For a random permutation, we expect that about $b/(e \cdot m!)$ of the b blocks contain m elements of P_i. Hence about 1/65 of all blocks have 4 or more elements of P_i. If the LDC is designed to protect against a 65δ-fraction of errors within one encoded block, then with overall error-fraction δ, the adversary has exactly enough noise to "destroy" all blocks containing 4 or more elements of P_i. The probability that our uniformly random j sits in such a "destroyed" block is about

$$\sum_{m \geq 4} \frac{m}{b} \frac{b}{e \cdot m!} = \frac{1}{e} \left(\frac{1}{3!} + \frac{1}{4!} + \cdots \right) \approx 0.08.$$

Hence if we set the error of the BMRV-structure to 1/10 and the error of the LDC to 1/100 (as above), then the total error probability for decoding x_i is less than 0.2 (of course we need to show that we can fix a π such that good decoding occurs for a good fraction of all $i \in [n]$). Another parameter that may be adjusted is the block size, which we here took to be $1000s$. Clearly, different tradeoffs between codelength, tolerable noise-level, and error probability are possible.

3. The Inner product problem

3.1. Noiseless case

Here we show bounds for Inner product, first for the noiseless case ($\delta = 0$).

Upper bound. Consider all strings z of weight at most $\lceil r/p \rceil$. The number of such z is $B(n, \lceil r/p \rceil) = \sum_{i=0}^{\lceil r/p \rceil} \binom{n}{i} \leq (epn/r)^{r/p}$. We define our codeword by writing down, for all z in lexicographic order, the inner product $x \cdot z$ mod 2. If we want to recover the inner product $x \cdot y$ for some y of weight at most r, we write $y = z_1 + \cdots + z_p$ for z_j's of weight at most $\lceil r/p \rceil$ and recover $x \cdot z_j$ for each $j \in [p]$, using one probe for each. Summing the results of the p probes gives $x \cdot y$ (mod 2). For $p = 1$ probes, the length is $B(n, r)$.

Lower bound. To prove a nearly-matching lower bound, we use Miltersen's technique of relating a data structure to a two-party communication game [Mil94]. We refer to [KN97] for a general introduction to communication complexity. Suppose Alice gets string $x \in \{0,1\}^n$, Bob gets string $y \in \{0,1\}^n$ of weight $\leq r$, and they need to compute $x \cdot y$ (mod 2) with bounded error probability and minimal communication between them. Call this communication problem $\mathrm{IP}_{n,r}$. Let $B(n,r) = \sum_{i=0}^{r} \binom{n}{i}$ be the size of Q, i.e., the number of possible queries y. For reasons of space we skip the proof of the following communication complexity lower bound, which may be found in the full version of this paper.

Theorem 3.1. *Every communication protocol for* $\mathrm{IP}_{n,r}$ *with worst-case (or even average-case) success probability* $\geq 1/2+\beta$ *needs at least* $\log(B(n,r))-2\log(1/2\beta)$ *bits of communication.*

Armed with this communication bound we lower bound data structure length:

Theorem 3.2. *Every* (p,ε)*-data structure for* $\mathrm{IP}_{n,r}$ *needs* $N \geq \frac{1}{2}2^{(\log(B(n,r))-2\log(1/(1-2\varepsilon))-1)/p}$.

Proof. We will use the data structure to obtain a communication protocol for $\mathrm{IP}_{n,r}$ that uses $p(\log(N)+1)+1$ bits of communication, and then invoke Theorem 3.1 to obtain the lower bound. Alice holds x, and hence $\phi(x)$, while Bob simulates the decoder. Bob starts the communication. He picks his first probe to the data structure and sends it over in $\log N$ bits. Alice sends back the 1-bit answer. After p rounds of communication, all p probes have been simulated and Bob can give the same output as the decoder would have given. Bob's output will be the last bit of the communication. Theorem 3.1 now implies $p(\log(N)+1)+1 \geq \log(B(n,r))-2\log(1/(1-2\varepsilon))$. Rearranging gives the bound on N. ∎

For fixed ε, the lower bound is $N=\Omega\left(B(n,r)^{1/p}\right)$. This is $\Omega((n/r)^{r/p})$, which (at least for small p) is not too far from the upper bound of approximately $(epn/r)^{r/p}$ mentioned above. Note that in general our bound on N is superpolynomial in n whenever $p=o(r)$. For instance, when $r=\alpha n$ for some constant $\alpha\in(0,1/2)$ then $N=\Omega(2^{nH(\alpha)/p})$, which is non-trivial whenever $p=o(n)$. Finally, note that the proof technique also works if Alice's messages are longer than 1 bit (i.e., if the code is over a larger-than-binary alphabet).

3.2. Noisy case

3.2.1. *Constructions for* Substring. One can easily construct error-correcting data structures for Substring, which also suffice for Inner product. Note that since we are recovering r bits, and each probe gives at most one bit of information, by information theory we need at least about r probes to the data structure. Our solutions below will use $O(r\log r)$ probes. View x as a concatenation $x=x^{(1)}\dots x^{(r)}$ of r strings of n/r bits each (we ignore rounding for simplicity), and define $\phi(x)$ as the concatenation of the Hadamard codes of these r pieces. Then $\phi(x)$ has length $N=r\cdot 2^{n/r}$.

If $\delta\geq 1/4r$ then the adversary could corrupt one of the r Hadamard codes by 25% noise, ensuring that some of the bits of x are irrevocably lost even when we allow the full N probes. However, if $\delta\ll 1/r$ then we can recover each bit x_i with small constant error probability by 2 probes in the Hadamard codeword where i sits, and with error probability $\ll 1/r$ using $O(\log r)$ probes. Hence we can compute $f(x,y)=x_y$ with error close to 0 using $p=O(r\log r)$ probes (or with $2r$ probes if $\delta\ll 1/r^2$). This also implies that *any* data structure problem where $f(x,q)$ depends on at most some fixed constant r bits of x, has an error-correcting data structure of length $N=r\cdot 2^{n/r}$, $p=O(r\log r)$, and that works if $\delta\ll 1/r$. Alternatively, we can take Efremenko's [Efr08] or Yekhanin's 3-probe LDC [Yek07], and just decode each of the r bits separately. Using $O(\log r)$ probes to recover a bit with error probability $\ll 1/r$, we recover the r-bit string x_y using $p=O(r\log r)$ probes even if δ is a constant independent of r.

3.2.2. *Constructions for* INNER PRODUCT. Going through the proof of [Yek07], it is easy to see that it allows us to compute the parity of any set of r bits from x using at most $3r$ probes with error ε, if the noise rate δ is at most $\varepsilon/(3r)$ (just add the results of the 3 probes one would make for each bit in the parity). To get error-correcting data structures even for small constant p (independent of r), we can adapt the polynomial schemes from [BIK05] to get the following theorem. The details are given in the full version of this paper.

Theorem 3.3. *For every $p \geq 2$, there exists a $(p, \delta, p\delta)$-error-correcting data structure for $\mathrm{IP}_{n,r}$ of length $N \leq p \cdot 2^{r(p-1)^2 n^{1/(p-1)}}$.*

For the $p = 2$ case, we get something simpler and better from the Hadamard code. This code, of length 2^n, actually allows us to compute $x \cdot y \pmod 2$ for any $y \in \{0,1\}^n$ of our choice, with 2 probes and error probability at most 2δ (just probe z and $y \oplus z$ for uniformly random $z \in \{0,1\}^n$ and observe that $(x \cdot z) \oplus (x \cdot (y \oplus z)) = x \cdot y$). Note that for $r = \Theta(n)$ and $p = O(1)$, even non-error-correcting data structures need length $2^{\Theta(n)}$ (Theorem 3.2). This is an example where error-correcting data structures are not significantly longer than the non-error-correcting kind.

4. Future work

Many questions are opened up by our model of error-correcting data structures, e.g.:

- There are plenty of other natural data structure problems, such as RANK, PREDECESSOR, versions of NEAREST NEIGHBOR etc. [Mil99]. What about the length-vs-probes tradeoffs for their error-correcting versions? The obvious approach is to put the best known LDC on top of the best known non-error-correcting data structures. This is not always optimal, though—for instance in the case of s-out-of-n MEMBERSHIP one can do significantly better, as we showed.
- It is often natural to assume that a memory cell contains not a bit, but some number from, say, a polynomial-size universe. This is called the *cell-probe* model [Yao81], in contrast to the *bit-probe* model we considered here. Probing a cell gives $O(\log n)$ bits at the same time, which can significantly improve the length-vs-probes tradeoff and is worth studying. Still, we view the bit-probe approach taken here as more fundamental than the cell-probe model. A p-probe cell-probe structure is a $O(p \log n)$-probe bit-probe structure, but not vice versa. Also, the way memory is addressed in actual computers in constant chunks of, say, 8 or 16 bits at a time, is closer in spirit to the bit-probe model than to the cell-probe model.
- Zvi Lotker suggested to me the following connection with distributed computing. Suppose the data structure is distributed over N processors, each holding one bit. Interpreted in this setting, an error-correcting data structure allows an honest party to answer queries about the encoded object while communicating with at most p processors. The answer will be correct with probability $1 - \varepsilon$, even if up to a δ-fraction of the N processors are faulty or even malicious (the querier need not know where the faulty/malicious sites are).

Acknowledgments

Thanks to Nitin Saxena for many useful discussions, to Harry Buhrman and Jaikumar Radhakrishnan for discussions about [BMRV00], to Zvi Lotker for the connection with distributed computation mentioned in Section 4, to Peter Bro Miltersen for a pointer to [JMM07], and to Gabriel Moruz for sending me a copy of that paper.

References

[AB96] Y. Aumann and M. Bender. Fault-tolerant data structures. In *Proceedings of 37th IEEE FOCS*, pages 580–589, 1996.

[BFF+07] G. Brodal, R. Fagerberg, I. Finocchi, F. Grandoni, G. Italiano, A. Jørgenson, G. Moruz, and T. Mølhave. Optimal resilient dynamic dictionaries. In *Proceedings of 15th ESA*, pages 347–358, 2007.

[BIK05] A. Beimel, Y. Ishai, and E. Kushilevitz. General constructions for information-theoretical Private Information Retrieval. *Journal of Computer and System Sciences*, 72(2):247–281, 2005.

[BMRV00] H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? *SIAM Journal on Computing*, 31(6):1723–1744, 2002.

[CIK+01] R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective private function evaluation with applications to private statistics. In *Proceedings of 20th ACM PODC*, pages 293–304, 2001.

[Efr08] K. Efremenko. 3-Query locally decodable codes of subexponential length. ECCC Report TR08–069, 2008.

[FGI07] I. Finocchi, F. Grandoni, and G. Italiano. Resilient search trees. In *Proceedings of 18th ACM-SIAM SODA*, pages 547–553, 2007.

[FI04] I. Finocchi and G. Italiano. Sorting and searching in the presence of memory faults (without redundancy). In *Proceedings of 36th ACM STOC*, pages 101–110, 2004.

[FKS84] M. Fredman, M. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[JMM07] A. G. Jørgenson, G. Moruz, and T. Mølhave. Resilient priority queues. In *Proceedings of 10th WADS*, volume 4619 of *Lecture Notes in Computer Science*, Springer, 2007.

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[KT00] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of 32nd ACM STOC*, pages 80–86, 2000.

[KW04] I. Kerenidis and R. de Wolf. Exponential lower bound for 2-query locally decodable codes via a quantum argument. *Journal of Computer and System Sciences*, 69(3):395–420, 2004.

[Mil94] P. B. Miltersen. Lower bounds for Union-Split-Find related problems on random access machines. In *Proceedings of 26th ACM STOC*, pages 625–634, 1994.

[Mil99] P. B. Miltersen. Cell probe complexity - a survey. Invited paper at *Advances in Data Structures* workshop. Available at Miltersen's homepage, 1999.

[MS77] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[RSV02] J. Radhakrishnan, P. Sen, and S. Venkatesh. The quantum complexity of set membership. *Algorithmica*, 34(4):462–479, 2002.

[Tre04] L. Trevisan. Some applications of coding theory in computational complexity. *Quaderni di Matematica*, 13:347–424, 2004.

[TS02] A. Ta-Shma. Storing information with extractors. *Information Processing Letters*, 83(5):267–274, 2002.

[vL98] J. H. van Lint. *Introduction to Coding Theory*. Springer, third edition, 1998.

[Yao81] A. C-C. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.

[Yek07] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. In *Proceedings of 39th ACM STOC*, pages 266–274, 2007.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 325–336
www.stacs-conf.org

FRAGMENTS OF FIRST-ORDER LOGIC OVER INFINITE WORDS
(EXTENDED ABSTRACT)

VOLKER DIEKERT [1] AND MANFRED KUFLEITNER [1]

[1] Universität Stuttgart, FMI, Universitätsstraße 38, D-70569 Stuttgart, Germany

Abstract. We give topological and algebraic characterizations as well as language theoretic descriptions of the following subclasses of first-order logic FO[<] for ω-languages: Σ_2, Δ_2, $\mathrm{FO}^2 \cap \Sigma_2$ (and by duality $\mathrm{FO}^2 \cap \Pi_2$), and FO^2. These descriptions extend the respective results for finite words. In particular, we relate the above fragments to language classes of certain (unambiguous) polynomials. An immediate consequence is the decidability of the membership problem of these classes, but this was shown before by Wilke [20] and Bojańczyk [2] and is therefore not our main focus. The paper is about the interplay of algebraic, topological, and language theoretic properties.

1. Introduction

The algebraic approach for fragments of first-order logic over finite words has been very fruitful. For example, a result of Wilke and Thérien is that FO^2 and Δ_2 have the same expressive power [14], where the latter class by definition denotes $\Sigma_2 \cap \Pi_2$. Further results are language theoretic and (very often decidable) algebraic characterizations of logical fragments, see e.g. [13] or [4] for surveys. Several results for finite words have been extended to other structures such as trees and other graphs, see [18] for a survey. For some characterizations over finite words, it has been shown that they cannot be generalized; e.g. over unranked trees, it turned out that FO^2 and Δ_2 are incomparable [1]. For infinite words, it is clear that the expressive power of FO^2 is not equal to Δ_2, since saying that letters a and b appear infinitely often, but c only finitely many times is FO^2-definable, but there is neither a Σ_2-formula nor a Π_2-formula specifying this language.

Our results deepen the understanding of first-order fragments over infinite words. A decidable characterization of the membership problem for FO^2 over infinite words has been given in the habilitation thesis of Wilke [20]. Recently, decidability for Σ_2 has been shown independently by Bojańczyk [2]. Language theoretic and decidable algebraic characterizations of the fragment Σ_1 and of its Boolean closure can be found in [8, 9].

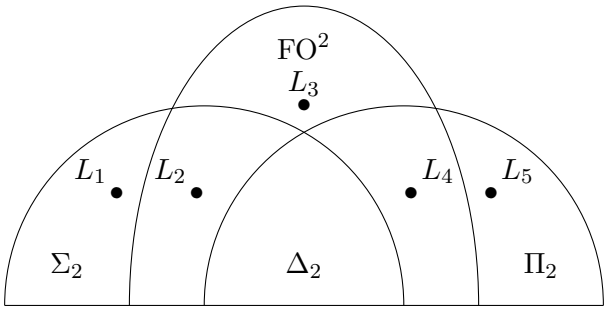
We introduce two generalizations of the usual Cantor topology for infinite words. One of our first results is a characterization of languages $L \subseteq \Gamma^\infty$ being Σ_2-definable in terms

Received by the editors December 16, 2008.
1998 ACM Subject Classification: F.4.1 Mathematical Logic, F.4.3 Formal Languages.
Key words and phrases: infinite words, regular languages, first-order logic, automata theory, semigroups, topology.

of a property of its syntactic monoid and by requiring that L is open in some alphabetic topology. Both properties are decidable.

Our second result is that a language is FO^2-definable if and only if its syntactic monoid is in the variety **DA**. (The result is surprising in the sense that it contradicts an explicit statement in [20]). Moreover, we show that FO^2-definability can be characterized by being closed in some further refined alphabetic topology and in terms of weak recognition by some monoid in **DA**. In particular, weak recognition and strong recognition do not coincide for the variety **DA**. This seems to be a new result as well. We also contribute a language theoretic characterization of FO^2 in terms of unambiguous polynomials with additional constraints on the letters which occur infinitely often.

Further main results of our paper are the characterization of $\mathrm{FO}^2 \cap \Sigma_2$ as the class of unambiguous polynomials and of Δ_2 in terms of unambiguous polynomials in some special form. In particular, it follows already from this description that Δ_2 is a strict subset of FO^2. Furthermore, we show that the equality of FO^2 and Δ_2 holds relativized to some fixed set of letters which occur infinitely often. If this set of letters is empty, we obtain the situation for finite words as a special case. Finally, we relate topological constructions such as *interior* and *closure* with membership in the fragments under consideration. Among other results, we are going to explain the following relations between the fragments FO^2, Σ_2, Π_2, and $\Delta_2 = \Sigma_2 \cap \Pi_2$:



Here $\Gamma = \{a, b, c\}$ and
L_1 = "there exists a factor ab"
$= \Gamma^* ab \Gamma^\infty$
L_2 = "finitely many a's"
L_3 = "finitely many a's and infinitely many b's" $= L_2 \cap L_4$
L_4 = "infinitely many b's"
L_5 = "there is no factor ab" $= \Gamma^\infty \setminus L_1$

It will turn out that L_4 is the closure of L_3 within some alphabetic topology, whereas L_2 is not the interior of L_3 since $L_3 \subsetneq L_2$. In fact, the interior of L_3 with respect to our topology is empty.

For basic notions on languages of infinite words we refer to standard references such as [8, 16].

2. Preliminaries

Words. Throughout, Γ is a finite alphabet, $A \subseteq \Gamma$ is a subset of the alphabet, u, v, w are finite words, and α, β, γ are finite or infinite words. If not specified otherwise, then in all examples we assume that Γ has three different letters a, b, c. By $u \le \alpha$ we mean that u is a (finite) prefix of α. By $\mathrm{alph}(\alpha)$ we denote the *alphabet* of α, i.e., the letters occurring in the sequence α. As usual, Γ^* is the free monoid of finite words over Γ. The neutral element is the empty word 1. If L is a subset of a monoid, then L^* is the submonoid generated by L. For $L \subseteq \Gamma^*$ we let $L^\omega = \{u_1 u_2 \cdots \mid u_i \in L \text{ for all } i \ge 1\}$ be the set of infinite products. We also let $L^\infty = L^* \cup L^\omega$. A natural convention is $1^\omega = 1$. Thus, $L^\infty = L^\omega$ if and only if $1 \in L$.

We write $\mathrm{im}(\alpha)$ for those letters in $\mathrm{alph}(\alpha)$ which have infinitely many different occurrences in α. The notation has been introduced in the framework of so called *complex traces*, see e.g. [6] for a detailed discussion of this concept. The notation $\mathrm{im}(\alpha)$ refers to the *imaginary part* and we adopt it here. A crucial role for us are sets of the form A^{im}, where, by definition, A^{im} is the set of words α such that $\mathrm{im}(\alpha) = A$. Note that $\Gamma^* = \emptyset^{\mathrm{im}}$. The set Γ^∞ is the disjoint union over all A^{im}.

Logic and regular sets. We assume that the reader is familiar with basic concepts in formal language theory. All languages L here can be assumed to be *regular*. The finite part $L \cap \Gamma^*$ can be assumed to be specified by some NFA and infinite part $L \cap \Gamma^\omega$ can be assumed to be specified by some Bchi automaton. We focus on regular languages which are given by first-order sentences in FO[$<$]. Thus, atomic predicates are $\lambda(x) = a$ and $x < y$ saying that position x in a word α is labeled with $a \in \Gamma$ and position x is less than y, respectively. By FO^2 we mean FO[$<$]-sentences which use at most two names x and y as variables or the class of languages specified by such formulas. Similarly, Σ_2 means FO[$<$]-sentences which are in prenex normal form and which start with a block of existential quantifiers, followed by a block of universal quantifiers and a Boolean combination of atomic formulas. A Π_2-formula means a negation of a Σ_2-formula. The notations Σ_2 and Π_2 refer also to the corresponding language classes. The class Δ_2 means the class of Σ_2-formulas which have an equivalent Π_2-formula. But the notion of equivalence depends on the set of models we use. If the models are finite words, then a result of Thérien and Wilke [14] states $\mathrm{FO}^2 = \Delta_2$. Moreover, FO^2 is the class of regular languages in Γ^* which are recognized by some finite monoid in the variety **DA** and a classical result of Schützenberger shows that **DA** also coincides with unambiguous polynomials [10]. We refer to [12, 4] for more background on the class **DA**. It is a class of finite monoids defined e.g. by equations of type $(xy)^\omega = (xy)^\omega y (xy)^\omega$. We recall that the class **DA** can also be defined by equations of the form $e = ese$ for all idempotents e (i.e., $e^2 = e$) and for all s generated by factors of e, see e.g. [17].

Saying that formulas are equivalent if they agree on all finite and infinite words changes the picture. This is actually the starting point of this work. So, in this paper models are finite and infinite words. We are mainly interested in infinite words, but it does no harm to include finite words, and this makes the situation more uniform and the results on finite words reappear as special cases. See e.g. Theorem 8.1 which means $\mathrm{FO}^2 = \Delta_2$ for finite words by choosing $A = \emptyset$. An important concept in this paper is topology.

3. The alphabetic topology and polynomials

We equip Γ^∞ with a refinement of the usual Cantor topology. The languages $u\Gamma^\infty$ form a basis of the Cantor topology. As we will see, topological information is crucial in our characterization results. We define the *alphabetic topology* by its basis, which is given by all sets of the form uA^∞. Thus, a set L is *open* if and only if for each $A \subseteq \Gamma$ there is a set of finite words $W_A \subseteq \Gamma^*$ such that $L = \bigcup W_A A^\infty$. By definition, a set is *closed*, if its complement is open; and it is *clopen*, if it is both open and closed. All sets A^∞ are clopen. A set A^{im} is not open unless $A = \emptyset$, it is not closed unless $A = \Gamma$.

Remark 3.1. The space Γ^∞ with the alphabetic topology is Hausdorff, but not compact, in general (in contrast to the Cantor topology). To see that it is not compact for $\Gamma = \{a, b\}$ note that $\Gamma^\infty = a^\omega \cup \Gamma^* b \Gamma^\infty$. The singleton set a^ω is clopen, but for no finite subset $F \subseteq \Gamma^*$ we have $\Gamma^\infty = a^\omega \cup Fb\Gamma^\infty$.

For a language L, its *closure* $\overline{L}$ is the intersection of all closed sets containing L. A word $\alpha \in \Gamma^\infty$ belongs to $\overline{L}$ if for all open subsets $U \subseteq \Gamma^\infty$ with $\alpha \in U$ we have $U \cap L \neq \emptyset$. The *interior* of L is the complement of the closure of its complement. For languages L and K we define the right quotient as a language of finite words by $L/K = \{u \in \Gamma^* \mid u\alpha \in L \text{ for some } \alpha \in K\}$. For $L \subseteq \Gamma^*$ we define

$$\overrightarrow{L} = \{\alpha \in \Gamma^\infty \mid \text{for every prefix } u \leq \alpha \text{ there exists } uv \leq \alpha \text{ with } uv \in L\}\,.$$

Proposition 3.2. *In the alphabetic topology we have* $\overline{A^{\mathrm{im}}} = \bigcup_{A \subseteq B} B^{\mathrm{im}}$ *and*

$$\overline{L} = \bigcup_{A \subseteq \Gamma} \left(\overrightarrow{L/A^\infty} \cap A^{\mathrm{im}}\right) = \bigcup_{A \subseteq \Gamma} \left(\overrightarrow{L/A^\infty} \cap \overline{A^{\mathrm{im}}}\right).$$

Corollary 3.3. *Given a regular language* $L \subseteq \Gamma^\infty$*, we can decide whether* L *is closed (open resp., clopen resp.).*

Actually, we have a more precise statement than pure decidability.

Theorem 3.4. *The following problem is* PSPACE*-complete:*
Input: A Bchi automaton $\mathcal{A}$ *with* $L(\mathcal{A}) \subseteq \Gamma^\omega$*.*
Question: Is the regular language $L(\mathcal{A})$ *closed?*

Remark 3.5. Neither languages of the form $\overrightarrow{L/A^\infty}$ nor $\overrightarrow{L/A^\infty} \cap \overline{A^{\mathrm{im}}}$ as in Proposition 3.2 need to be closed. Indeed, let $A = \{a\}$, $B = \{a, b\}$, and $L = a^*(ab)^*ba^\omega$. Then $L/A^\infty = a^*(ab)^*ba^*$ and L/B^∞ is the set of all finite prefixes of words in L. We have $\overrightarrow{L/A^\infty} = a^*(ab)^*ba^\infty$ and $\overrightarrow{L/A^\infty} \cap \overline{A^{\mathrm{im}}} = a^*(ab)^*ba^\omega = L$. The language $\overrightarrow{L/A^\infty}$ is open but neither $\overrightarrow{L/A^\infty}$ nor $\overrightarrow{L/A^\infty} \cap \overline{A^{\mathrm{im}}}$ is closed in the alphabetic topology, because $(ab)^\omega$ belongs to both closures. We have $\overrightarrow{L/B^\infty} = a^*(ab)^*ba^\infty \cup a^*(ab)^\omega$ and $\overrightarrow{L/B^\infty} \cap B^{\mathrm{im}} = a^*(ab)^\omega$. Both sets are closed. Actually, $\overline{L} = L \cup a^*(ab)^\omega$ in the alphabetic topology. Finally note that $\overline{L}$ is not closed in the Cantor topology since $a^\omega \notin \overline{L}$. Remember that a basis of the Cantor topology are the sets of the form $u\Gamma^\infty$.

Frequently we apply the closure operator to polynomials. A *polynomial* is a finite union of monomials. A *monomial* (of *degree* k) is a language of the form $A_1^*a_1 \cdots A_k^*a_kA_{k+1}^\infty$ with $a_i \in \Gamma$ and $A_i \subseteq \Gamma$. In particular, $A_1^*a_1 \cdots A_k^*a_k$ is a monomial with $A_{k+1} = \emptyset$. The set A^* is a polynomial since $A^* = \emptyset^\infty \cup \bigcup_{a \in A} A^*a$. It is not hard to see that polynomials are closed under intersection. Thus, $A_1^*a_1 \cdots A_k^*a_kA_{k+1}^*$ is in our language a polynomial, but not a monomial unless $A_{k+1} = \emptyset$. A monomial $P = A_1^*a_1 \cdots A_k^*a_kA_{k+1}^\infty$ is *unambiguous* if for every $\alpha \in P$ there exists a unique factorization $\alpha = u_1a_1 \cdots u_ka_k\beta$ such that $u_i \in A_i^*$ and $\beta \in A_{k+1}^\infty$. A polynomial is *unambiguous* if it is a finite union of unambiguous monomials.

It follows from the definition of the alphabetic topology that polynomials are open. Actually, it is the coarsest topology with this property. The crucial observation is that we have a syntactic description of the closure of a polynomial as a finite union of other polynomials. For later use we make a more precise statement.

Lemma 3.6. *Let* $P = A_1^*a_1 \cdots A_k^*a_kA_{k+1}^\infty$ *be a monomial and* $L = P \cap B^{\mathrm{im}}$ *for some* $B \subseteq A_{k+1}$*. Then the closure of* L *is given by*

$$\bigcup_{\{a_i, \ldots, a_k\} \cup B \subseteq A \subseteq A_i} A_1^*a_1 \cdots A_{i-1}^*a_{i-1}A_i^\infty \cap A^{\mathrm{im}}.$$

Proof. First consider an index i with $1 \leq i \leq k+1$ such that $\{a_i, \ldots, a_k\} \cup B \subseteq A \subseteq A_i$. Let $\alpha \in A_1^* a_1 \cdots A_{i-1}^* a_{i-1} A_i^\infty \cap A^{\mathrm{im}}$. We have to show that α is in the closure of L. Let $\alpha = u\beta$ with $u \in A_1^* a_1 \cdots A_{i-1}^* a_{i-1} A_i^*$ and $\beta \in A^\infty \cap A^{\mathrm{im}}$. We show that $uA^\infty \cap L \neq \emptyset$. Choose some $\gamma \in B^\infty \cap B^{\mathrm{im}}$. As $B \subseteq A_{k+1}$ holds by hypothesis, we see that $ua_i \cdots a_k \gamma \in P$, and hence $ua_i \cdots a_k \gamma \in uA^\infty \cap L$.

Let now $\alpha \in \overline{L}$ and write $\alpha \in uv_1 \cdots v_{k+1} A^\infty \cap A^{\mathrm{im}}$ with $\mathrm{alph}(v_j) = A$. There exists $\gamma \in A^\infty$ such that $uv_1 \cdots v_{k+1}\gamma \in P \cap B^{\mathrm{im}}$. This implies $B \subseteq A$. Since $uv_1 \cdots v_{k+1}\gamma \in A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty$ there are some $1 \leq i, j \leq k+1$ such that $uv_1 \cdots v_{j-1}$ belongs to $A_1^* a_1 \cdots A_{i-1}^* a_{i-1} A_i^* \cap A^{\mathrm{im}}$, $v_j \in A_i^*$, and $v_{j+1} \cdots v_{k+1}\gamma \in A_i^* a_i \cdots A_k^* a_k A_{k+1}^\infty \cap A^\infty$. Therefore $\{a_i, \ldots, a_k\} \subseteq A \subseteq A_i$, too. It follows that $\alpha \in A_1^* a_1 \cdots A_{i-1}^* a_{i-1} A_i^\infty \cap A^{\mathrm{im}}$. ∎

4. Recognizability by finite monoids

By M we denote a finite monoid. We always assume that M is equipped with a partial order $\leq$ being compatible with the multiplication, i.e., $u \leq v$ implies $sut \leq svt$ for all $s, t, u, v \in M$. If not specified otherwise, we may choose $\leq$ to be the identity relation.

For an idempotent element $e \in M$ we define $M_e = \{s \in M \mid e \in MsM\}^*$. By definition, M_e is a submonoid of M. If M is generated by Γ, then M_e is generated by $\{a \in \Gamma \mid e \in MaM\}$. We can think of this set as the maximal alphabet of the idempotent e. We say that an idempotent e is *locally top* (*locally bottom*, resp.) if $ese \leq e$ ($ese \geq e$, resp.) for all $s \in M_e$. By **DA** we denote the class of finite monoids such that $ese = e$ for all idempotents $e \in M$ and all $s \in M_e$. More information about this variant to define **DA** can be found in [17].

Let $L \subseteq \Gamma^\infty$ be a language. The *syntactic preorder* $\leq_L$ over Γ^* is defined as follows. We let $u \leq_L v$ if for all $x, y, z \in \Gamma^*$ we have both implications:

$$xvyz^\omega \in L \;\Rightarrow\; xuyz^\omega \in L \qquad \text{and} \qquad x(vy)^\omega \in L \;\Rightarrow\; x(uy)^\omega \in L.$$

Let us recall that $1^\omega = 1$. Two words $u, v \in \Gamma^*$ are syntactically equivalent, written as $u \equiv_L v$, if both $u \leq_L v$ and $v \leq_L u$. This is a congruence and the congruence classes $[u]_L = \{v \in \Gamma^* \mid u \equiv_L v\}$ form the *syntactic monoid* $\mathrm{Synt}(L)$ of L. The preorder $\leq_L$ on words induces a partial order $\leq_L$ on congruence classes, and $(\mathrm{Synt}(L), \leq_L)$ becomes an ordered monoid. It is a well-known classical result that the syntactic monoid of a regular language $L \subseteq \Gamma^\infty$ is finite, see e.g. [8, 16]. Moreover, in this case L can be written as a finite union of languages of type $[u]_L\,[v]_L^\omega$ where $u, v \in \Gamma^*$ with $uv \equiv_L u$ and $v^2 \equiv_L v$.

Now, let $h : \Gamma^* \to M$ be any surjective homomorphism onto a finite ordered monoid M and let $L \subseteq \Gamma^\infty$. If the reference to h is clear, then we denote by $[s]$ the set of finite words $h^{-1}(s)$ for $s \in M$. The following notations are used:

- $(s, e) \in M \times M$ is a *linked pair*, if $se = s$ and $e^2 = e$.
- h *weakly recognizes* L, if
$$L = \bigcup \{[s][e]^\omega \mid (s, e) \text{ is a linked pair and } [s][e]^\omega \subseteq L\}$$
- h *strongly recognizes* L (or simply *recognizes* L), if
$$L = \bigcup \{[s][e]^\omega \mid (s, e) \text{ is a linked pair and } [s][e]^\omega \cap L \neq \emptyset\}$$
- L is *downward closed (on finite prefixes)* for h, if $[s][e]^\omega \subseteq L$ implies $[t][e]^\omega \subseteq L$ for all $s, t, e \in M$ where $t \leq s$.

Lemma 4.1. *Let $L \subseteq \Gamma^\infty$ be a regular language and let $h_L : \Gamma^* \to \mathrm{Synt}(L)$ be its syntactic homomorphism. Then for all $s, t, e, f \in M$ such that $t \leq s$, $f \leq e$, and $[s][e]^\omega \subseteq L$ we have $[t][f]^\omega \subseteq L$. In particular, L is downward closed (on finite prefixes) for h_L.*

Proof. Let $u \in [s]$, $x \in [e]$ and let $v \in [t]$, $y \in [f]$. Now, $ux^\omega \in L$ implies $vx^\omega \in L$, which in turn implies $vy^\omega \in L$. Since L is regular, h_L strongly recognizes L, and we obtain $[t][f]^\omega \subseteq L$ because $vy^\omega \in [t][f]^\omega \cap L$. ∎

For lack of space and in order to avoid too much machinery we do not treat ω-semigroups [9, 19] in this extended abstract. However, let us define $tf^\omega \leq_L se^\omega$ for linked pairs by the implication:

$$[s][e]^\omega \subseteq L \;\Rightarrow\; [t][f]^\omega \subseteq L.$$

With this notation we can give an algebraic characterization of being open.

Lemma 4.2. *A regular language $L \subseteq \Gamma^\infty$ is open in the alphabetic topology if and only if for all linked pairs (s, e), (t, f) of $M = \mathrm{Synt}(L)$ with $t, f \in M_e$ we have $stf^\omega \leq_L se^\omega$.*

5. The fragment Σ_2

By a (slight extension of a) result of Thomas [15] on ω-languages we know that a language $L \subseteq \Gamma^\infty$ is definable in Σ_2 if and only if L is a polynomial. However, this statement alone does not yield decidability. It turns out that we obtain decidability by a combination of an algebraic and a topological criterion. This decidability result has also been shown by Bojańczyk [2] using different techniques. We know that polynomials are open. Therefore, we concentrate on algebra.

Lemma 5.1. *If $L \subseteq \Gamma^\infty$ is a polynomial, then all idempotents of $\mathrm{Synt}(L)$ are locally top.*

Theorem 5.2. *Let $L \subseteq \Gamma^\infty$ be a regular language. The following assertions are equivalent:*

1. *L is Σ_2-definable.*
2. *L is a polynomial.*
3. *L is open in the alphabetic topology and all idempotents of $\mathrm{Synt}(L)$ are locally top.*
4. *The syntactic monoid $M = \mathrm{Synt}(L)$ and the syntactic order $\leq_L$ satisfy:*
   (a) *For all linked pairs (s, e), (t, f) with $t, f \in M_e$ we have $stf^\omega \leq_L se^\omega$.*
   (b) *$e = e^2$ and $s \in M_e$ implies $ese \leq_L e$.*
5. *The following three conditions hold for some homomorphism $h : \Gamma^* \to M$ which weakly recognizes L:*
   (a) *L is open in the alphabetic topology.*
   (b) *All idempotents of M are locally top.*
   (c) *L is downward closed (on finite prefixes) for h.*

Proof. "1 ⇔ 2": This is a slight modification of a result by Thomas [15].
"2 ⇒ 3": By definition, polynomials are open in the alphabetic topology. In Lemma 5.1 it has been shown that all idempotent elements are locally top.
"3 ⇔ 4": The equivalence of L being open and "4a" is Lemma 4.2. Property "4b" is the definition of all elements being locally top.
"4 ⇒ 5": Let $h = h_L$ be the syntactic homomorphism onto the syntactic monoid $M = \mathrm{Synt}(L)$. Applying Lemma 4.2, property "5a" follows from "4a" and "5b" trivially follows from "4b". The condition "5c" holds for $\mathrm{Synt}(L)$ by Lemma 4.1.

"5 ⇒ 2": Consider $\alpha \in L$ with $\mathrm{im}(\alpha) = A$. By "5a" the language L is open. Hence, there exists a prefix u of α such that $\alpha \in uA^\infty \subseteq L$. From the case of finite words and the hypothesis "5b" on M, we know that $P = \{v \in \Gamma^* \mid h(v) \leq h(u)\}$ is a polynomial. We can assume that all monomials in P end with a letter. We define the polynomial $P_\alpha = PA^\infty$. Clearly, $L \subseteq \bigcup \{P_\alpha \mid \alpha \in L\}$ and this union is finite since M is finite. It remains to show that $P_\alpha \subseteq L$ for $\alpha \in L$. Let $v \in P$ and $\beta \in A^\infty$. We know $u\beta \in L$ and there exists a linked pair (s, e) such that $u\beta \in [s][e]^\omega \subseteq L$. Now, there exists $w\gamma = \beta$ such that $uw \in [s]$ and $\gamma \in [e]^\omega$. By definition of P, we have $h(v) \leq h(u)$ and therefore $t = h(vw) \leq h(uw) = s$. It follows $v\beta = vw\gamma \in [t][e]^\omega \subseteq L$ by "5c". This shows $P_\alpha \subseteq L$ and thus $L = \bigcup \{P_\alpha \mid \alpha \in L\}$. ∎

Corollary 5.3. *It is decidable whether a regular language is Σ_2-definable.*

Remark 5.4. An ω-language $L \subseteq \Gamma^\omega$ is Σ_2-definable, if $L = \{\alpha \in \Gamma^\omega \mid \alpha \models \varphi\}$ for some $\varphi \in \Sigma_2$. This is equivalent with $L \cup \Gamma^*$ being Σ_2-definable as a subset of Γ^∞. Thus, the decidability of Corollary 5.3 transfers to ω-regular languages.

6. Two variable first-order logic

Etessami, Vardi, and Wilke have given a characterization of FO^2 in terms of unary temporal logic [5]. In the same paper, they considered the satisfiability problem for FO^2. In this section, we continue the study of FO^2 over infinite words.

The following lemma can be proved essentially in the same way as for finite words. The result is also (implicitly) stated in the habilitation thesis of Wilke [20].

Lemma 6.1. *Let $L \subseteq \Gamma^\infty$ be FO^2-definable. Then the syntactic monoid* $\mathrm{Synt}(L)$ *is in* **DA**.

A set like A^{im} is FO^2-definable, but it is neither open nor closed in the alphabetic topology, in general. Therefore, we need a refinement of the alphabetic topology. As a basis for the *strict alphabetic topology* we take all sets of the form $uA^\infty \cap A^{\mathrm{im}}$. Thus, more sets are open (and closed) than in the alphabetic topology. Another way to define the strict alphabetic topology is to say that it is the coarsest topology on Γ^∞ where all sets of the form $A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty \cap B^{\mathrm{im}}$ are open. The strict alphabetic topology is not used outside this section, but it is essential here in order to prove the converse of Lemma 6.1.

Lemma 6.2. *If $L \subseteq \Gamma^\infty$ is strongly recognized by some homomorphism $h : \Gamma^* \to M \in$ **DA**, then L is clopen in the strict alphabetic topology.*

Proof. Since h also strongly recognizes $\Gamma^\infty \setminus L$ as well, it is enough to show that L is open. Let $\alpha \in L$ with $\alpha \in [s][e]^\omega$ for some linked pair (s, e) and let $A = \mathrm{im}(\alpha)$. We show that $[s]A^\infty \cap A^{\mathrm{im}} \subseteq L$. Indeed, let $\beta \in [s]A^\infty \cap A^{\mathrm{im}}$. Then we have $\beta = uv\gamma$ with $h(u) = s$, $h(v) = r$, $\gamma \in [f]^\omega$ where $v \in A^*$, $\mathrm{alph}(\gamma) = \mathrm{im}(\gamma) = A$, and (r, f) is a linked pair. Since $M \in$ **DA**, we obtain $s = se = serfe = srfe$ and $efe = e$ and $fef = f$. Since h strongly recognizes L, we can compute as follows:

$$\beta \in [sr][f]^\omega = [sr][fef]^\omega = [srfe][efe]^\omega = [s][e]^\omega \subseteq L$$

In particular, $\beta \in L$. ∎

Lemma 6.3. *If L is closed in the strict alphabetic topology and if L is weakly recognized by some homomorphism $h : \Gamma^* \to M \in \mathbf{DA}$, then L is a finite union of languages $A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty \cap A_{k+1}^{\mathrm{im}}$, where each $A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty$ is an unambiguous monomial.*

Proof. Let $\alpha \in L$. Write $\alpha = u\beta$ with $\beta \in A^\infty \cap A^{\mathrm{im}}$ for some $A \subseteq \Gamma$. There is a linked pair (s, e) with $\alpha \in [s][e]^\omega \subseteq L$ and we may assume $h(u) = s$ and $\beta \in [e]^\omega$. For $A = \emptyset$ we have $[s] \subseteq L$ and, using our knowledge about the finite case, we may include $[s]$ in our finite union of unambiguous polynomials. Therefore, let $A \neq \emptyset$. We may choose an unambiguous monomial $P = A_1^* a_1 \cdots A_k^* a_k \subseteq [s]$ such that $u \in P$ and each last position of every letter $a \in \{a_1, \ldots, a_k\} \cup A_1 \cup \cdots \cup A_k$ occurs explicitly as some a_j in the expression P. Note that $[s]$ is a finite union of such monomials. Moreover, we may assume that $uv \in P$ for infinitely many prefixes $v \leq \beta$. Each such uv can uniquely be written as $uv = v_1 a_1 \cdots v_k a_k$ with $v_i \in A_i^*$. This yields a vector in $\mathbb{N}^k$ by $(|v_1 a_1|, |v_1 a_1 v_2 a_2|, \ldots, |v_1 a_1 \cdots v_k a_k|)$ for every $uv \in P$. By Dickson's Lemma [3], we may assume that this vector is in no component decreasing when v gets longer. Hence (after removing finitely many v's) we may assume there is some i such that $|v_1 a_1 \cdots v_i a_i|$ is constant and $|v_1 a_1 \cdots v_i a_i v_{i+1} a_{i+1}|$ is strictly increasing. It follows that we may assume $\{a_{i+1}, \ldots, a_k\} \subseteq \mathrm{alph}(v_{i+1}) = A \subseteq A_{i+1}$. In particular, $\alpha \in A_1^* a_1 \cdots A_i^* a_i A^\infty \cap A^{\mathrm{im}}$. It is clear that this expression is unambiguous.

It remains to show $A_1^* a_1 \cdots A_i^* a_i A^\infty \cap A^{\mathrm{im}} \subseteq L$. Consider $u'\gamma$ with $u' \in A_1^* a_1 \cdots A_i^* a_i$ and $\gamma \in A^\infty \cap A^{\mathrm{im}}$. Since L is closed, it is enough to show that $u'\gamma$ belongs to the closure of L in the strict alphabetic topology. Choose any prefix $w \leq \gamma$. It is enough to show that $u' w A^\infty \cap A^{\mathrm{im}} \cap L \neq \emptyset$. Let $z \in \Gamma^*$ with $\mathrm{alph}(z) = A$ and $h(z) = e$. Since $w \in A^* \subseteq A_{i+1}^*$, we have $u' w a_{i+1} \cdots a_k \in P \subseteq [s]$. Hence $u' w a_{i+1} \cdots a_k z^\omega \in [s][e]^\omega \subseteq L$. ∎

Lemma 6.4. *Every language A^{im} and every unambiguous monomial $A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty$ is FO^2-definable.*

Theorem 6.5. *Let $L \subseteq \Gamma^\infty$. The following assertions are equivalent:*

1. *L is FO^2-definable.*
2. *L is regular and $\mathrm{Synt}(L) \in \mathbf{DA}$.*
3. *L is strongly recognized by some homomorphism $h : \Gamma^* \to M \in \mathbf{DA}$.*
4. *L is closed in the strict alphabetic topology and L is weakly recognized by some homomorphism $h : \Gamma^* \to M \in \mathbf{DA}$.*
5. *L is a finite union of sets of the form $A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty \cap A_{k+1}^{\mathrm{im}}$, where each language $A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty$ is an unambiguous monomial.*

Proof. "1 ⇒ 2": First-order definable languages are regular; $\mathrm{Synt}(L) \in \mathbf{DA}$ by Lemma 6.1. "2 ⇒ 3": Trivial, since $\mathrm{Synt}(L)$ strongly recognizes L. "3 ⇒ 4": Strong recognition implies weak recognition; closure in the strict alphabetic topology follows by Lemma 6.2. "4 ⇒ 5": Lemma 6.3. "5 ⇒ 1": Lemma 6.4. ∎

Recall that if a language $L \subseteq \Gamma^\infty$ is weakly recognizable by a finite monoid, then it is also strongly recognizable by a finite monoid. The same holds for aperiodic monoids, but Theorem 6.5 suggests that this fails for **DA**. Indeed, we have the following example.

Example 6.6. Let $\Gamma = \{a, b, c\}$. Consider the congruence of finite index such that each class $[u]$ is defined by the set of words v where u and v agree on all suffixes of length at most 2. The quotient monoid of Γ^* by this congruence is in **DA**. Let $L = [ab]^\omega = (\Gamma^* ab)^\omega$. Then, by definition, L is weakly recognizable in **DA**. But L is the language of all α which contain infinitely many factors of the form ab. This is however not closed for the strict alphabetic

topology since $(acb)^\omega \notin L$, but $(acb)^\omega$ belongs to the strict alphabetic closure of L since every open set U with $(acb)^\omega \in U$ contains some $(acb)^m(cab)^\omega$ and $[(acb)^m(cab)] = [ab]$ for all $m \geq 0$.

7. Unambiguous polynomials and the fragments $\mathbf{FO^2 \cap \Sigma_2}$ and $\mathbf{FO^2 \cap \Pi_2}$

Theorem 7.1. *Let $L \subseteq \Gamma^\infty$. The following assertions are equivalent:*

(1) *L is both FO^2-definable and Σ_2-definable.*
(2) *L is FO^2-definable and open in the alphabetic topology.*
(3) *L is a finite union of unambiguous monomials of the form $A_1^* a_1 \cdots A_k^* a_k A_{k+1}^\infty$.*
(4) *L is the interior of some FO^2-definable language.*

Theorem 7.2. *Let $L \subseteq \Gamma^\infty$ be a regular language. The following assertions are equivalent:*

(1) *L is both FO^2-definable and Π_2-definable.*
(2) *L is FO^2-definable and closed in the alphabetic topology.*
(3) *L is the closure of some FO^2-definable language.*

Theorem 7.2 is not fully satisfactory since we do not have any direct characterization in terms of polynomials. We might wish that if L is closed (and $L \in \Pi_2 \cap \mathrm{FO}^2$), then it is a finite union of languages $K \cap B^{\mathrm{im}}$ where each $K \cap B^{\mathrm{im}}$ is closed. But this is not true: Let $L = \Gamma^* a \cup \Gamma^\omega$, then L is closed and in $\Pi_2 \cap \mathrm{FO}^2$, but cannot be written in this form because $L = \Gamma^* a$ is not closed. We also note that the closure of a language L in $\mathrm{FO}^2 \cap \Sigma_2$ needs not to be in Δ_2. A counter-example is the language $L = \Gamma^* abc$. By Lemma 3.6, the closure of L is $\overline{L} = L \cup \Gamma^{\mathrm{im}}$ which is not Σ_2-definable.

8. The fragment $\mathbf{\Delta_2 = \Sigma_2 \cap \Pi_2}$

For finite words we have the well-known theorem that FO^2-definability is equivalent to Δ_2-definability. However, this does not transfer to ω-words where Δ_2 forms a proper subclass of FO^2. Consider $L = \{a, b\}^{\mathrm{im}}$, then L is neither open nor closed, in general. Hence $L \in \mathrm{FO}^2 \setminus (\Sigma_2 \cup \Pi_2)$. The result for finite words is therefore somewhat misleading. The correct translation for the general case is:

Theorem 8.1. *For all $A \subseteq \Gamma$ the following assertions are equivalent:*

(1) *$L \cap A^{\mathrm{im}}$ is FO^2-definable.*
(2) *There are languages $L_\sigma \in \mathrm{FO}^2 \cap \Sigma_2$ and $L_\pi \in \mathrm{FO}^2 \cap \Pi_2$ such that*
$$L \cap A^{\mathrm{im}} = L_\sigma \cap A^{\mathrm{im}} = L_\pi \cap A^{\mathrm{im}}.$$
(3) *There are languages $L_\sigma \in \Sigma_2$ and $L_\pi \in \Pi_2$ such that*
$$L \cap A^{\mathrm{im}} = L_\sigma \cap A^{\mathrm{im}} = L_\pi \cap A^{\mathrm{im}}.$$

Note that we cannot expect that $L_\sigma = L_\pi$ in the statement above, because L_σ is open and L_π is closed. Hence, a language in Δ_2 must be clopen. The first step for a convenient characterization on Δ_2 is therefore a description of clopen unambiguous monomials.

Lemma 8.2. *Let $P = A_1^* a_1 \cdots A_k^* a_k A^\infty$ be an unambiguous monomial. The following assertions are equivalent:*

1. *There is no $1 \le i \le k$ such that $\{a_i, \ldots, a_k\} \subseteq A_i$.*
2. *P is closed in the alphabetic topology.*
3. *P is clopen in the alphabetic topology.*

Proof. "1 ⇒ 2": For a moment let $A_{k+1} = A$. By Lemma 3.6 we know that the closure of P is:

$$\bigcup_{\{a_i,\ldots,a_k\} \subseteq B \subseteq A_i} A_1^* a_1 \cdots A_{i-1}^* a_{i-1} (A_i^\infty \cap B^{\mathrm{im}}).$$

Since there is no $\{a_i, \ldots, a_k\} \subseteq A_i$ for $1 \le i \le k$, we see that this union is just P itself. Therefore, P is closed. "2 ⇒ 3": is clear, because P is open. "3 ⇒ 1": Assume by contradiction that $\{a_i, \ldots, a_k\} \subseteq A_i$ for some $1 \le i \le k$. We have $a_1 \cdots a_{i-1}(a_i \cdots a_k)^m \in P$ for all $m \ge 1$. As P is closed we see $a_1 \cdots a_{i-1}(a_i \cdots a_k)^\omega \in P$ and hence $\{a_i, \ldots, a_k\} \subseteq A$. But this is a contradiction to the fact that P is unambiguous since $\{a_i, \ldots, a_k\} \subseteq A_i \cap A$ implies that $a_1 \cdots a_{i-1}(a_i \cdots a_k)^2 \in P$ has two different factorizations. ∎

Lemma 8.3. *Let $L \subseteq \Gamma^\infty$ be a closed polynomial. For every unambiguous monomial $P = A_1^* a_1 \cdots A_k^* a_k A^\infty \subseteq L$ there exist closed unambiguous monomials $Q_1, \ldots, Q_\ell$ such that $P \subseteq Q_1 \cup \cdots \cup Q_\ell \subseteq L$, i.e., there exists a finite covering of P with closed unambiguous monomials in L.*

Theorem 8.4. *Let $L \subseteq \Gamma^\infty$. The following assertions are equivalent.*

1. *L is Δ_2-definable.*
2. *L is FO^2-definable and L is clopen in the alphabetic topology.*
3. *L is a finite union of unambiguous closed monomials $A_1^* a_1 \cdots A_k^* a_k A^\infty$, i.e., there is no $1 \le i \le k$ such that $\{a_i, \ldots, a_k\} \subseteq A_i$.*
4. *L is regular, $\mathrm{Synt}(L) \in \mathbf{DA}$, and for all linked pairs (s, e), (t, f) with $s \mathcal{R} t$ (i.e., there exist $x, y \in \mathrm{Synt}(L)$ such that $s = tx$ and $t = sy$) we have*

$$[s][e]^\omega \subseteq L \;\Leftrightarrow\; [t][f]^\omega \subseteq L.$$

Proof. "1 ⇒ 2": By Theorem 5.2 and its dual version for Π_2, we see that $\mathrm{Synt}(L) \in \mathbf{DA}$ and that L is clopen in the alphabetic topology. From Theorem 6.5 it follows that L is FO^2-definable. "2 ⇒ 3": By Theorem 7.1, L is a finite union of unambiguous monomials. Property "3" now follows by Lemma 8.3 and Lemma 8.2. "3 ⇒ 1": Theorem 7.1 and Theorem 7.2.

"2 ⇒ 4": By Theorem 6.5, we see that $\mathrm{Synt}(L) \in \mathbf{DA}$. Suppose $[s][e]^\omega \subseteq L$ and let $s = tx$ and $t = sy$. Since L is closed we see that $[s][eyfx]^\omega \subseteq L$ and by strong recognition we conclude $[t][fxey]^\omega \subseteq L$. Let $A = \bigcup \{\mathrm{alph}(v) \mid v \in [f]\}$. Since L is open and by strong recognition, there exists $r \in \mathbb{N}$ such that $[t][fxey]^r A^\infty \subseteq L$. Moreover, $t = tfxey$ and thus, $[t]A^\infty \subseteq L$. In particular, $[t][f]^\omega \subseteq L$ because $[f] \subseteq A^*$.

"4 ⇒ 2": Definability in FO^2 follows by Theorem 6.5. By symmetry, it suffices to show that L is open. Let $\alpha \in [s][e]^\omega \subseteq L$ for some linked pair (s, e) and write $\alpha = u\beta$ with $u \in [s]$ and $\beta \in [e]^\omega \cap A^\infty \cap A^{\mathrm{im}}$ for some $A \subseteq \Gamma$. Let $v \le \beta$ be a prefix such that $v \in [e]$ and $\mathrm{alph}(v) = \mathrm{alph}(\beta)$. We want to show $uvA^\infty \subseteq L$. Consider $uv\gamma \in \Gamma^\infty$ where $\gamma \in A^\infty$. We have $uv\gamma \in [t][f]^\omega$ for some linked pair (t, f). Let $v' \le \gamma$ such that $uvv' \in [t]$. Since $\mathrm{Synt}(L) \in \mathbf{DA}$ we have $vv'v \in [e]$ and $s = t \cdot h(v)$. Together with $t = s \cdot h(v')$ it follows $s \mathcal{R} t$ and by "4" we obtain $uv\gamma \in [t][f]^\omega \subseteq L$. ∎

9. Summary

We gave language-theoretic, algebraic and topological characterizations for several first-order fragments over infinite words. Since FO^2 and Δ_2 have the same expressive power only when restricted to some fixed set of letters occurring infinitely often (Thm. 8.1), the picture becomes more complex. By Pol we denote the language class of polynomials, UPol are unambiguous polynomials, and *restricted* UPol is a proper subclass of UPol. All of the below-mentioned algebraic properties are decidable, since the syntactic monoid of a regular language is effectively computable [8, 16]. Together with the PSPACE-completeness of the problem whether a language is closed in the alphabetic topology (Thm. 3.4), this yields decidability of the membership problem for the respective first-order fragments as a corollary. Decidability was shown before by Wilke [20] for FO^2 and by Bojańczyk [2] for Σ_2. Characterizations for the fragment Σ_1 and its Boolean closure over infinite words (using topological notions based on the Cantor topology) are due to Pin [9]; see also [8].

Logic	**Languages**	**Algebra**		**Topology**	
Σ_2	Pol	$eM_ee \leq e$	+	open (alphabetic)	Thm. 5.2
FO^2	UPol + A^{im}	strong **DA**			Thm. 6.5
		weak **DA**	+	closed (strict alphabetic)	
$\mathrm{FO}^2 \cap \Sigma_2$	UPol	**DA**	+	open (alphabetic)	Thm. 7.1
$\mathrm{FO}^2 \cap \Pi_2$		**DA**	+	closed (alphabetic)	Thm. 7.2
Δ_2	restricted UPol	**DA**	+	clopen (alphabetic)	Thm. 8.4

10. Outlook and open problems

By definition, Σ_1-definable languages are open in the Cantor topology. We introduced an alphabetic topology such that Σ_2-definable languages are open in this topology. Therefore, an interesting question is whether it is possible to extend this topological approach to higher levels of the first-order alternation hierarchy. To date, even over finite words no decidable characterization of the Boolean closure of Σ_2 is known. In case that a decidable criterion is found, it might lead to a decidable criterion for infinite words simply by adding a condition of the form "clopen in some appropriate topology". Another possible way to generalize our approach might be combinations of algebraic and topological characterizations for fragments with successor predicate suc such as $\mathrm{FO}^2[<, \mathrm{suc}]$ or $\Sigma_2[<, \mathrm{suc}]$. A characterization of those languages which are weakly recognizable by monoids in **DA** is also open.

Acknowledgements

We thank the anonymous referees for many useful suggestions and we thank Luc Dartois from the ENS Cachan for helpful discussions during his internship in Stuttgart and for his firm conviction that FO^2 should coincide with **DA**.

References

[1] M. Bojańczyk. Two-way unary temporal logic over trees. In *LICS'07*, pages 121–130. IEEE Computer Society, 2007.
[2] M. Bojańczyk. The common fragment of ACTL and LTL. In *Foundations of Software Science and Computational Structures, 11th International Conference, FoSSaCS 2008, Proceedings*, volume 4962 of *Lecture Notes in Computer Science*, pages 172–185. Springer-Verlag, 2008.
[3] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913.
[4] V. Diekert, P. Gastin, and M. Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(3):513–548, June 2008. Special issue DLT 2007.
[5] K. Etessami, M. Y. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279–295, 2002.
[6] P. Gastin and A. Petit. Infinite traces. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 11, pages 393–486. World Scientific, Singapore, 1995.
[7] A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE Computer Society, 1972.
[8] D. Perrin and J.-É. Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, Amsterdam, 2004.
[9] J.-É. Pin. Positive varieties and infinite words. In C. Lucchesi and A. Moura, editors, *Latin'98*, volume 1380 of *Lecture Notes in Computer Science*, pages 76–87. Springer-Verlag, 1998.
[10] M. P. Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13:47–75, 1976.
[11] A. P. Sistla, M. Y. Vardi, and P. L. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2-3):217–237, 1987.
[12] P. Tesson and D. Thérien. Diamonds are Forever: The Variety DA. In G. M. dos Gomes Moreira da Cunha, P. V. A. da Silva, and J.-É. Pin, editors, *Semigroups, Algorithms, Automata and Languages, Coimbra (Portugal) 2001*, pages 475–500. World Scientific, 2002.
[13] P. Tesson and D. Thérien. Logic meets algebra: The case of regular languages. *Logical Methods in Computer Science*, 3(1):1–37, 2007.
[14] D. Thérien and Th. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC*, pages 234–240, 1998.
[15] W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25:360–376, 1982.
[16] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990.
[17] P. Weil. Some results on the dot-depth hierarchy. *Semigroup Forum*, 46:352–370, 1993.
[18] P. Weil. Algebraic recognizability of languages. In *Mathematical Foundations of Computer Science 2004*, volume 3153 of *Lecture Notes in Computer Science*, pages 149–174. Springer, Berlin, 2004.
[19] Th. Wilke. An Eilenberg theorem for ∞-languages. In *ICALP'91*, volume 510 of *Lecture Notes in Computer Science*, pages 588–599. Springer-Verlag, 1991.
[20] Th. Wilke. *Classifying Discrete Temporal Properties*. Habilitationsschrift, Universität Kiel, April 1998.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 337–348
www.stacs-conf.org

UNDECIDABLE PROPERTIES OF LIMIT SET DYNAMICS OF CELLULAR AUTOMATA

PIETRO DI LENA [1] AND LUCIANO MARGARA [1]

[1] Department of Computer Science, University of Bologna, Mura Anteo Zamboni 7, Bologna, Italy

E-mail address, P. Di Lena: dilena@cs.unibo.it
E-mail address, L. Margara: margara@cs.unibo.it

Abstract. Cellular Automata (CA) are discrete dynamical systems and an abstract model of parallel computation. The limit set of a cellular automaton is its maximal topological attractor. A well know result, due to Kari, says that all nontrivial properties of limit sets are undecidable. In this paper we consider properties of limit set dynamics, i.e. properties of the dynamics of Cellular Automata restricted to their limit sets. There can be no equivalent of Kari's Theorem for limit set dynamics. Anyway we show that there is a large class of undecidable properties of limit set dynamics, namely all properties of limit set dynamics which imply stability or the existence of a unique subshift attractor. As a consequence we have that it is undecidable whether the cellular automaton map restricted to the limit set is the identity, closing, injective, expansive, positively expansive, transitive.

Introduction

Cellular Automata (CA) are discrete dynamical systems and, at the same time, an abstract model of parallel computation. Every cellular automaton has a finite description in terms of a finite block mapping called *local rule*. A general problem for CA is to determine what are the properties which are algorithmically decidable/undecidable given the local rule.

The limit set Ω_F of a cellular automaton $(A^{\mathbb{Z}}, F)$ is the set of all configurations which occur after arbitrarily long iterates of the CA map, i.e. $x \in \Omega_F$ if and only if $\forall n \in \mathbb{N}, F^{-n}(x) \neq \emptyset$. The limit set is the maximal topological attractor of a cellular automaton (then it is always nonempty and closed) and it is fundamental to understand the long-term behavior of such systems. Kari's Theorem [Kari94] says that all nontrivial properties of limit sets are undecidable. This implies, for example, that we cannot decide algorithmically if some given configuration is in the limit set or not and we cannot even decide if some given word is contained in some configuration of the limit set. Kari's undecidability theorem uniquely regards properties of the configurations contained in the limit set, but it does not include properties of the dynamics of Cellular Automata restricted to their limit set. The motivation of this work is to try to understand what are the undecidable properties of the limit set dynamics, i.e. properties of the dynamical systems (Ω_F, F). It is easy to find simple examples of nontrivial decidable properties of $F : \Omega_F \to \Omega_F$ which imply that Kari's

1998 ACM Subject Classification: Theory of Computation, Computation by Abstract Devices.
Key words and phrases: Cellular Automata, Undecidability, Symbolic Dynamics.

Theorem cannot be extended to whole limit set dynamics. Anyway, we can show that there is a large and interesting class of properties of $F : \Omega_F \to \Omega_F$ which are undecidable. For instance, we show that any property of limit set dynamics which implies stability or the existence of a unique subshift attractor is undecidable. Stated in another way, we obtain that any decidable property of limit set dynamics must be a property of some unstable cellular automaton with at least two subshift attractors. As a consequence we show that it is not possible to decide algorithmically whether the cellular automaton map restricted to the limit set is the identity, closing, injective, expansive, positively expansive and transitive.

The paper is organized as follows. In Section 1 we provide the basic background in Symbolic Dynamics and Cellular Automata needed to understand the rest of the paper. In Section 2 we formally define what properties of limit sets are and we show some preliminary results. In Section 3 we discuss our main results. Section 4 is devoted to concluding remarks.

1. Preliminaries

1.1. Symbolic Dynamics

In this section we review only those notions which are strictly necessary to understand our proofs. See [LM95] for a complete introduction to Symbolic Dynamics.

Let A be a finite alphabet with at least two elements. We denote by A^n the set of words of length n over A, by $A^* = \cup_{n\in\mathbb{N}} A^n$ the set of words over A and by $A^{\mathbb{Z}}$ the set of doubly infinite sequences $(x_i)_{i\in\mathbb{Z}}$ of symbols $x_i \in A$. We denote by $x_{[i,j]} \in A^{j-i+1}$ the subword $x_i x_{i+1} ... x_j$. We use the shortcut $w \sqsubset x$ to say that $w \in A^+$ is a subword of $x \in A^{\mathbb{Z}}$.

Define a *metric* d on $A^{\mathbb{Z}}$ by $d(x, y) = 2^{-n}$ where $n = min\{|i| \mid x_i \neq y_i\}$. The set $A^{\mathbb{Z}}$ endowed with metric d is a compact metric space. For $u \in A^*$ and $i \in \mathbb{Z}$, denote by $[u]_i = \{x \in A^{\mathbb{Z}} \mid x_{[i,i+|u|-1]} = u\}$ a *cylinder set*. For a lighter notation, we will refer to the cylinder set $[u]_i$ simply by $[u]$. A cylinder set is a clopen (closed and open) set in $A^{\mathbb{Z}}$. Every clopen set in $A^{\mathbb{Z}}$ is a finite union of cylinder sets.

The *shift map* $\sigma : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is defined by $\sigma(x)_i = x_{i+1}$. The shift map is continuous and biiective on $A^{\mathbb{Z}}$. The dynamical system $(A^{\mathbb{Z}}, \sigma)$ is called *full shift*. A *shift space* or *subshift* is a non-empty closed subset $\Sigma \subseteq A^{\mathbb{Z}}$ which is strongly shift invariant, i.e. $\sigma(\Sigma) = \Sigma$. We will usually denote the *shift dynamical system* (Σ, σ) simply with Σ. A subshift Σ is a *zero-dimensional space*, i.e. for every two different points $x, y \in \Sigma$ there exists disjoint clopen sets $U, V \subset \Sigma$ such that $x \in U, y \in V$.

We denote by $\mathcal{L}_n(\Sigma) = \{w \in A^n \mid \exists x \in \Sigma, w \sqsubset x\}$ the set of words of length n of the subshift Σ. The *language* of Σ is defined by $\mathcal{L}(\Sigma) = \cup_{n\in\mathbb{N}} \mathcal{L}_n(\Sigma)$. Any subshift Σ is completely determined by the set of its *forbidden words* $A^* \setminus \mathcal{L}(\Sigma)$. A *shift of finite type* (SFT) is a subshift which can be defined by a *finite* set of forbidden words. Let Σ be a subshift on alphabet A. We denote by $\Sigma_k = \{x \in A^{\mathbb{Z}} \mid \forall i \in \mathbb{Z}, x_{[i,i+k)} \in \mathcal{L}_k(\Sigma)\}$ the *SFT approximation of order* $k > 0$ of Σ. Note that $\forall k > 0, \Sigma \subseteq \Sigma_k$ and that Σ_k is a SFT since it is defined by the finite set of forbidden words $A^k \setminus \mathcal{L}_k(\Sigma)$. If Σ is a SFT then there exists some $k > 0$ such that $\forall k' \geq k, \Sigma = \Sigma_{k'}$. We say that the least such $k > 0$ is the *order* of Σ. A generalization of SFTs are *sofic shifts*. A subshift S is sofic if and only if its language $\mathcal{L}(S)$ is *regular*. A subshift Σ is *mixing* if there exists $n > 0$ such that for all clopen sets $U, V \subseteq \Sigma$, $\sigma^n(U) \cap V \neq \emptyset$.

Let Σ_1, Σ_2 be subshifts. A *factor map* $F : \Sigma_1 \to \Sigma_2$ is a continuous, onto, σ-commuting mapping. A factor map is actually a *block code*, i.e. F is induced by some *k-block mapping* $f : \mathcal{L}_1^k(\Sigma_1) \to \mathcal{L}_1(\Sigma_2)$ where $k > 0$. The mixing and sofic properties are preserved under factor maps.

A factor map F is *right-closing* if $x, y \in \Sigma, x_{(-\infty,i]} = y_{(-\infty,i]}$ and $F(x) = F(y)$ imply $x = y$. The definition of left-closing is equivalent. By using a simple compactness argument it is possible to prove that closing is equivalent to the following condition: $\exists n > 0$ such that $\forall x, y \in \Sigma, \forall i \in \mathbb{Z}$ if $x_{[i,i+n)} = y_{[i,i+n)}$ and $F(x)_{[i,i+2n]} = F(y)_{[i,i+2n]}$ then $x_{i+n} = y_{i+n}$. The closing property imposes strong constraint on the mapping. For example, it is possible to prove that if Σ is a mixing SFT and $F : \Sigma \to \Sigma$ is continuous, σ-commuting and closing then F is onto, i.e. $F(\Sigma) = \Sigma$.

An endomorphism $F : \Sigma \to \Sigma$ is *positively expansive* if there exists $\epsilon > 0$ such that for all distinct $x, y \in \Sigma$ there exists $n \in \mathbb{N}$ such that $d(F^n(x), F^n(y)) > \epsilon$. If F is invertible then it is *expansive* if there exists $\epsilon > 0$ such that for all distinct $x, y \in \Sigma$ there exists $n \in \mathbb{Z}$ such that $d(F^n(x), F^n(y)) > \epsilon$. Both expansive and positively expansive endomorphisms of subshifts must be closing. The map F is *transitive*, if for any nonempty open sets $U, V \subseteq \Sigma$ there exists $n \in \mathbb{N}$ such that $F^{-n}(U) \cap V \neq \emptyset$. Both expansive and positively expansive endomorphisms of mixing SFT are transitive.

1.2. Cellular Automata

One-dimensional *Cellular Automata* (CA) are endomorphisms of full shifts. We denote CA by pairs $(A^{\mathbb{Z}}, F)$ where $F : A^{\mathbb{Z}} \to A^{\mathbb{Z}}$ is some continuous and σ-commuting function. The *global rule* F is a $(2r+1)$-block map, i.e. there exists some *local rule* $f : A^{2r+1} \to A$ of *radius* $r \geq 0$ such that

$$\forall x \in A^{\mathbb{Z}}, F(x)_i = f(x_{i-r}, ..., x_{i+r}).$$

It is sometimes useful to extend the local rule to the finite-block mapping

$$f^* : A^k \to A^{k-2r} \text{ for every } k \geq 2r+1,$$

such that

$$f^*(x_1, ..., x_k) = f(x_1, ..., x_{2r+1}) f(x_2, ..., x_{2r+2}) .. f(x_{k-2r}, ..., x_k).$$

Our investigation regards properties of the *limit behavior* of Cellular Automata. To understand the limit behavior the concept of *attractor* is fundamental. An attractor is a nonempty closed set which attracts the orbits of its neighboring points.

Definition 1.1. Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton. The *ω-limit* of a set $U \subseteq A^{\mathbb{Z}}$ with respect to F is defined by $\omega_F(U) = \cap_{n>0} \overline{\cup_{m>n} F^m(U)}$.

When it is clear from the context, we will denote the ω-limit simply with ω. In zero-dimensional spaces the following two definitions of attractors are equivalent.

Definition 1.2. Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton. A nonempty closed set $Y \subseteq A^{\mathbb{Z}}$ such that $F(Y) = Y$ is an *attractor* of $(A^{\mathbb{Z}}, F)$

1. if $\forall \epsilon > 0, \exists \delta > 0$ such that $\forall x \in A^{\mathbb{Z}}$

$$d(x, Y) < \delta \implies \forall n \in \mathbb{N}, d(F^n(x), Y) < \epsilon \text{ and } \lim_{n \to \infty} d(F^n(x), Y) = 0.$$

2. if and only if $Y = \omega(U)$ where U is a clopen F-invariant set, i.e. $F(U) \subseteq U$.

A useful property of attractors is that every neighborhood of an attractor contains a clopen F-invariant set whose ω-limit is the attractor itself. We show the proof for completeness. We first need a general lemma.

Lemma 1.3. *Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton and let $U, V \subseteq A^{\mathbb{Z}}$ be clopen sets. Assume that $\forall x \in U, \exists n_x \in \mathbb{N}$ such that $F^{n_x}(x) \in V$. Then there exists $n \in \mathbb{N}$ such that $\forall x \in U, \exists n_x \leq n, F^{n_x}(x) \in V$.*

Proof. For $i \in \mathbb{N}$ define $X_i = \{x \in U \mid \forall j \leq i, F^j(x) \notin V\}$. Since U, V are clopen it follows that for every $i \in \mathbb{N}, X_i$ is clopen and $X_i \supseteq X_{i+1}$. Assume that for every $i \in \mathbb{N}, X_i \neq \emptyset$ then, by compactness, $X = \cap_{i\in\mathbb{N}} X_i$ is nonempty which implies that there exists $x \in U$ such that $\forall i \in \mathbb{N}, F^i(x) \notin V$ contradicting the hypothesis. ∎

Proposition 1.4. *Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton and let $Y \subseteq A^{\mathbb{Z}}$ be an attractor. Then for every $\epsilon > 0$ there is an F-invariant clopen set $U \subseteq \mathcal{B}_\epsilon(Y)$ such that $\omega(U) = Y$.*

Proof. For $\epsilon > 0$, denote $Y_\epsilon = \mathcal{B}_\epsilon(Y)$. Note that for every $\epsilon > 0$, Y_ϵ is a clopen set. Choose some $\epsilon > 0$. By definition, there is some $0 < \delta < \epsilon$ such that

$$x \in Y_\delta \implies \forall n, F^n(x) \in Y_\epsilon \text{ and } \lim_{n\to\infty} d(F^n(x), Y) = 0.$$

Choose some $0 < \epsilon_0 < \delta$ then there is some $0 < \delta_0 < \epsilon_0$ such that

$$x \in Y_{\delta_0} \implies \forall n, F^n(x) \in Y_{\epsilon_0} \text{ and } \lim_{n\to\infty} d(F^n(x), Y) = 0.$$

If $x \in Y_\delta$ then $\lim_{n\to\infty} d(F^n(x), Y) = 0$ so there is some $n_x \in \mathbb{N}$ such that $F^{n_x}(x) \in Y_{\delta_0}$. By Lemma 1.3, there is some $n \in \mathbb{N}$ such that for every $x \in Y_\delta, \exists n_x \leq n, F^{n_x}(x) \in Y_{\delta_0}$ then $\forall x \in Y_\delta, F^n(x) \in Y_{\epsilon_0} \subseteq Y_\delta$. We obtained that there is some $n \in \mathbb{N}$ such that $F^n(Y_\delta) \subseteq Y_\delta$ then Y_δ is F^n-invariant. We now define a clopen set $U \subseteq Y_\epsilon$ which is F-invariant.

Let $x \in Y_\delta$, since $F^n(Y_\delta) \subseteq Y_\delta$ and Y_δ is clopen, there is a word $w \sqsubset x$ such that $[w] \subseteq Y_\delta$ and $[(f^*)^n(w)] \subseteq Y_\delta$ (where the length of $(f^*)^n(w)$ is greater than 0). In particular, since Y_δ is the union of a finite collection of cylinders, there is a finite set of words $w_1^0, ..., w_{k_0}^0$ such that $Y_\delta = [w_1^0] \cup ... \cup [w_{k_0}^0]$ and $[(f^*)^n(w_i^0)] \subseteq Y_\delta$ for $1 \leq i \leq k_0$. By considering iterates of f^* on such words we can obtain a sequence of clopen sets $U_0, U_1, .., U_n$ such that $F(U_j) \subseteq U_{j+1}$. Set $U_0 = Y_\delta = [w_1^0] \cup ... \cup [w_{k_0}^0]$ and define the clopen set $U_1 = \cup_{i=1}^{k_0}([f^*(w_i^0)] \cap Y_\epsilon) = [w_1^1] \cup ... \cup [w_{k_1}^1]$. Note that for every $i \in [0, k_0]$ we have $F([w_i^0]) \subseteq [f^*(w_i^0)], F([w_i^0]) \subseteq Y_\epsilon$ and $[f^*(w_i^0)] \cap Y_\epsilon$ is clopen. Then $F(U_0) \subseteq U_1 \subseteq Y_\epsilon$. Iterating for $j \in [1, n]$ we obtain the sequence of clopen sets $U_j = \cup_{i=1}^{k_{j-1}}([f^*(w_i^{j-1})] \cap Y_\epsilon) = [w_1^j] \cup ... \cup [w_{k_j}^j]$ such that $F(U_{j-1}) \subseteq U_j \subseteq Y_\epsilon$. Now define $U = \cup_{j=0}^n U_j$. We have that $U \subseteq Y_\epsilon$ is clopen, $F(U) \subseteq U$ and $\omega(U) = Y$. ∎

In the context of Cellular Automata, a particular class of attractors are those attractors which are also subshifts.

Definition 1.5. Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton. A nonempty closed set $Y \subseteq A^{\mathbb{Z}}$ is a *subshift attractor* if it is an attractor and if $\sigma(Y) = Y$.

The following two propositions characterize subshift attractors of CA.

Definition 1.6. Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton. We say that a clopen and F-invariant set $U \subseteq A^{\mathbb{Z}}$ is *spreading* if there exists some $k > 0$ such that $F^k(U) \subseteq \sigma^{-1}(U) \cap U \cap \sigma(U)$.

Proposition 1.7. [FK07] *Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton and let $U \subseteq A^{\mathbb{Z}}$ be a clopen and F-invariant set. Then $\omega(U)$ is a subshift attractor if and only if U is spreading.*

Proposition 1.8. [FK07] *Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton and let $U \subseteq A^{\mathbb{Z}}$ be a clopen F-invariant spreading set. Then there exists a mixing SFT $\Sigma \subseteq U$ with the following properties:*

- $F(\Sigma) \subseteq \Sigma$
- $W = \{[w] \mid w \in \mathcal{L}_k(\Sigma), k \text{ is the order of } \Sigma\}$ *is clopen and F-invariant*
- $\omega(\Sigma) = \omega(W) = \omega(U)$.

A cellular automaton has at least one attractor which is called *limit set*.

Definition 1.9. Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton. The *limit set* of $(A^{\mathbb{Z}}, F)$ is defined by $\Omega_F = \cap_{i=0}^{\infty} F^i(A^{\mathbb{Z}}) = \omega_F(A^{\mathbb{Z}})$.

Note that a configuration $x \in A^{\mathbb{Z}}$ is in the limit set if and only if for every $n \in \mathbb{N}$, $F^{-n}(x) \neq \emptyset$. The limit set is a subshift attractor and it is also the *maximal attractor*, i.e. every other attractor is contained in the limit set. The limit set can be the *unique attractor*. In particular, if the map is transitive on the limit set then it is the unique attractor (the converse is not true). An attractor is a *minimal attractor* if it does not contain any proper subset which is an attractor. A unique attractor is both maximal and minimal. There is a very simple class of minimal attractors. We say that a state $s \in A$ is *spreading* if the local rule has the property $f(x_1, ..., x_{2r+1}) = s$ if $\exists x_i = s$. If a cellular automaton has a *spreading state* s then the clopen set $[s]$ is F-invariant and spreading and $\omega([s]) = \{...sss....\}$ is a minimal subshift attractor.

Cellular Automata limit sets received great attention. Here we review just some basic facts. One question which is still not well understood concerns the class of subshifts which can be limit sets of CA (see, for example, [Hurd90, Maass95]). The most immediate distiction is between limit sets of stable and unstable CA. A cellular automaton $(A^{\mathbb{Z}}, F)$ is called *stable* if there exists some $n \in \mathbb{N}$ such that $F^n(A^{\mathbb{Z}}) = \Omega_F$. It is called *unstable* otherwise. The limit sets of stable CA are mixing sofic shifts since they are factors of full shifts. There are sofic subshifts which are limit sets of unstable CA but no limit set of unstable CA can be a SFT [Hurd90]. It is actually unknown whether a subshift can be the limit set of both a stable and of an unstable CA. The simplest example of limit set subshift is the subshift consisting of just one configuration. A cellular automaton whose limit set is a single configuration is called *nilpotent*. If a cellular automaton is nilpotent then the unique configuration in the limit set must be fixed by both σ and F then the automaton must be stable.

2. Properties of limit sets

An important aspect of CA is that they can be enumerated. Every cellular automaton is described by its local rule. Local rules are defined by a finite amount of information and, in particular, for any fixed radius and cardinality of the alphabet there are only finitely many possible CA local rules.

Choose some enumeration function for CA local rules. We denote by $\#(A^{\mathbb{Z}}, F) \in \mathbb{N}$ the *rule number* associated to $(A^{\mathbb{Z}}, F)$. A *property* $\mathcal{P}$ of CA is a collection of CA rule numbers. A property is called *trivial* if either all CA have such property or none has. A property $\mathcal{P}$ is *decidable* whether there exists some algorithm such that, for any given $(A^{\mathbb{Z}}, F)$, it always

computes if either $\#(A^{\mathbb{Z}}, F) \in \mathcal{P}$ or $\#(A^{\mathbb{Z}}, F) \notin \mathcal{P}$. A subclass of CA properties are, in particular, properties of the limit sets.

Definition 2.1. A property $\mathcal{P}$ is a *property of limit sets* if and only if the following condition holds: if $\#(A^{\mathbb{Z}}, F) \in \mathcal{P}$ and $(B^{\mathbb{Z}}, G)$ is a cellular automaton such that $\Omega_F = \Omega_G$ then $\#(B^{\mathbb{Z}}, G) \in \mathcal{P}$.

Nilpotency is a property of limit sets. While a property of limit sets is always a property of CA, the converse is not always true. For example, surjectivity is not a property of the limit sets since it is easy to construct a surjective CA and a not surjective CA which have the same limit set. For example, let $(A^{\mathbb{Z}}, F)$ be a surjective CA and let $(B^{\mathbb{Z}}, G)$ be such that $B = A \cup \{b\}$ (with $b \notin A$) and $\forall x \in B^{\mathbb{Z}}, G(x) = F(x')$ where x' is obtained by substituting every occurrence of b in x with the symbol $a \in A$. Then G is not surjective and $G(B^{\mathbb{Z}}) = A^{\mathbb{Z}}$.

Decidability questions about properties of the limit sets received great attention. One of the most important undecidability results, due to Kari, is the following one.

Theorem 2.2. [Kari92] *Nilpotency is undecidable for CA.*

Nilpotency remains undecidable also under the additional condition of a spreading state. Nilpotency is the basis to prove the undecidability of most of the undecidable properties of CA. In particular, Kari showed that (the problem to decide) nilpotency is the easiest problem among all decision problems on the limit sets.

Theorem 2.3. [Kari94] *Every nontrivial property of CA limit sets is undecidable.*

For example, by Theorem 2.3, every nontrivial property which regards the language $\mathcal{L}(\Omega_F)$ is undecidable. Kari's Theorem does not concern properties of the dynamics of CA on the limit set. Here we investigate decidability questions about properties of *limit set dynamical systems* or properties of *limit set dynamics*.

Definition 2.4. A property $\mathcal{P}$ is a *property of limit set dynamics* if and only if the following condition holds: if $\#(A^{\mathbb{Z}}, F) \in \mathcal{P}$ and $(B^{\mathbb{Z}}, G)$ is a cellular automaton such that $\Omega_F = \Omega_G$ and $F|\Omega_F = G|\Omega_G$ then $\#(B^{\mathbb{Z}}, G) \in \mathcal{P}$.

Note that, by definition, properties of limit sets are properties of limit set dynamics while the converse is not true. It is evident that we cannot have the equivalent of Theorem 2.3 for limit set dynamics. In fact, it is easy to find nontrivial properties of the limit set dynamical systems which are decidable. Consider, for example, the set of decidable properties $\mathcal{P}_n = \{\#(A^{\mathbb{Z}}, F) \mid \exists x \in A^{\mathbb{Z}}, F^n(x) = x\}$. Since every F-periodic point is contained in the limit set, all $\mathcal{P}_n$ are properties of the limit set dynamics.

In the following section we will show that there is a large class of undecidable properties of the limit set dynamics. In particular, our main result concerns properties of stable CA and properties of CA which have a unique subshift attractor. To conclude this section we show some results related to these properties.

Proposition 2.5. *It is undecidable whether a cellular automaton has a unique subshift attractor.*

Proof. Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton with a spreading state $s \in A$. The clopen set $[s]$ is F-invariant, spreading and $\omega([s]) = \{...ssss...\}$. If we could decide whether a cellular automaton has a unique subshift attractor the we could decide if $\omega([s])$ is the unique attractor of $(A^{\mathbb{Z}}, F)$ and then we could decide if $(A^{\mathbb{Z}}, F)$ is nilpotent or not. ∎

Proposition 2.6. [Kari94] *It is undecidable whether a cellular automaton is stable.*

Proof. Assume that we can decide if some cellular automaton is stable or not. We show that it is possible to decide nilpotency. Let $(A^{\mathbb{Z}}, F)$ be cellular automaton. If $(A^{\mathbb{Z}}, F)$ is not stable then it is not nilpotent. If it is stable then there exist some $n \in \mathbb{N}$ such that $F^n(A^{\mathbb{Z}}) = \Omega_F$. Then is sufficient to compute all forward images of $A^{\mathbb{Z}}$ until we reach the limit set Ω_F and then check if it is a singleton. ∎

It is open the question whether stability is a property of limit sets (i.e. it is unknown whether there is a subshift which can be limit set both of a stable and of an unstable cellular automaton). We can show that stability is a property of limit set dynamics. This implies that, even if there exists a subshift which is both the limit set of some stable and of some unstable CA, then the dynamics of such automata on their limit sets must be distinct.

Proposition 2.7. *Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton. Assume that there is a cellular automaton $(B^{\mathbb{Z}}, G)$ such that $\Omega_F = \Omega_G$ and $F|_{\Omega_F} = G|_{\Omega_F}$. Then $(B^{\mathbb{Z}}, G)$ is stable if and only if $(A^{\mathbb{Z}}, F)$ is stable.*

Proof. Let r be the maximum between the radius of $(A^{\mathbb{Z}}, F)$ and the radius of $(B^{\mathbb{Z}}, G)$. By Proposition 1.4 and Proposition 1.7, there is a clopen, F-invariant spreading set $U \subseteq \mathcal{B}_{2^{-r}}(\Omega_F)$ such that $\omega_F(U) = \Omega_F$. By Proposition 1.8, there is a mixing SFT $\Sigma \subset U \subseteq \mathcal{B}_{2^{-r}}(\Omega_F)$ such that $F(\Sigma) \subseteq \Sigma$ and $\omega_F(\Sigma) = \Omega_F$. We show that $\mathcal{L}_{2r+1}(\Sigma) = \mathcal{L}_{2r+1}(\Omega_F)$. Since $\Omega_F \subseteq \Sigma$ it is clear that $\mathcal{L}_{2r+1}(\Sigma) \supseteq \mathcal{L}_{2r+1}(\Omega_F)$. Assume that $\mathcal{L}_{2r+1}(\Sigma) \not\subseteq \mathcal{L}_{2r+1}(\Omega_F)$. Then there must be a configuration $x \in \Sigma$ such that $x_{[-r,r]} \notin \mathcal{L}_{2r+1}(\Omega_F)$ but this would imply that $x \notin \mathcal{B}_{2^{-r}}(\Omega_F)$ which is a contradiction. Then we have $\mathcal{L}_{2r+1}(\Sigma) = \mathcal{L}_{2r+1}(\Omega_F) = \mathcal{L}_{2r+1}(\Omega_G)$ which implies that $F|_\Sigma = G|_\Sigma$ and $\Sigma \subseteq B^{\mathbb{Z}}$. To conclude the proof it is sufficient to show that there exist $n, m \in \mathbb{N}$ such that $F^n(A^{\mathbb{Z}}) \subseteq \Sigma$ and $G^m(A^{\mathbb{Z}}) \subseteq \Sigma$ which would imply that $(A^{\mathbb{Z}}, F)$ and $(B^{\mathbb{Z}}, G)$ are both stable if and only if (Σ, F) is stable and are both unstable otherwise.

Let W be the clopen set as defined in Proposition 1.8. Since $\omega_F(A^{\mathbb{Z}}) = \Omega_F$, by compactness, we have that for every $x \in A^{\mathbb{Z}}$ there exists $n_x \in \mathbb{N}$ such that $F^{n_x}(x) \in W$. Then, since W is F-invariant, by Lemma 1.4, there exists $n \in \mathbb{N}$ such that $F^n(A^{\mathbb{Z}}) \subseteq W$. Then, for every $i \in \mathbb{Z}$, it must be $F^n(\sigma^i(x)) \in W$ which implies that $F^n(x) \in \Sigma$. We obtained that there exists $n \in \mathbb{N}$ such that $F^n(A^{\mathbb{Z}}) \subseteq \Sigma$.

Let t be the order of Σ and let $k \in \mathbb{N}$ be such that $2k + 1 \geq t$. By using the same argument above, we can show that there exists a mixing SFT $\Sigma' \subset \mathcal{B}_{2^{-k}}(\Omega_G)$ such that $\omega_G(\Sigma') = \Omega_G$ and such that $F^m(B^{\mathbb{Z}}) \subseteq \Sigma'$ for some $m \in \mathbb{N}$. We just need to show that $\Sigma' \subseteq \Sigma$ to obtain that there exists $m \in \mathbb{N}$ such that $F^m(B^{\mathbb{Z}}) \subseteq \Sigma$. Denote with Ω_{2k+1} the SFT approximation of order $2k + 1$ of Ω_G. By using the same argument above, we have $\mathcal{L}_{2k+1}(\Sigma') = \mathcal{L}_{2k+1}(\Omega_G) = \mathcal{L}_{2k+1}(\Omega_{2k+1})$ so Σ' is of order $t' \geq 2k + 1$. Since Σ is of order $t \leq 2k + 1$ and $\mathcal{L}_t(\Sigma) \supseteq \mathcal{L}_t(\Omega_G)$ it follows that $\Sigma' \subseteq \Omega_{2k+1} \subseteq \Sigma$. ∎

3. Undecidable properties of limit set dynamics

In this section we show that there is a large class of properties of the dynamics on the limit set which are not decidable. In particular we show that are undecidable all properties of the limit set dynamics which are properties of stable CA or properties of CA with a unique subshift attractor.

Definition 3.1. Let $\mathcal{P}$ be a property of CA. We say that $\mathcal{P}$ is a *stable property* if $\forall \#(A^{\mathbb{Z}}, F) \in \mathcal{P}$, $(A^{\mathbb{Z}}, F)$ is stable.

Definition 3.2. Let $\mathcal{P}$ be a property of CA. We say that $\mathcal{P}$ is a *unique subshift attractor property* if $\forall \#(A^{\mathbb{Z}}, F) \in \mathcal{P}$, Ω_F is the unique subshift attractor of $(A^{\mathbb{Z}}, F)$.

To prove the undecidability of stable properties of limit set dynamics we need a preliminary result. We don't know if a not nilpotent cellular automaton with a spreading state must be unstable. Anyway, by a simple construction, given a cellular automaton with a spreading state, we can build a new cellular automaton with a spreading state which is nilpotent (then stable) if and only if the old one is nilpotent and it is unstable otherwise.

Lemma 3.3. *Let $(A^{\mathbb{Z}}, F)$ be a CA with a spreading state. Then it is possible to construct a CA $(B^{\mathbb{Z}}, G)$ with a spreading state such that $(B^{\mathbb{Z}}, G)$ is nilpotent if and only if $(A^{\mathbb{Z}}, F)$ is nilpotent and $(B^{\mathbb{Z}}, G)$ is unstable otherwise.*

Proof. Let $s \in A$ and $r \in \mathbb{N}$ be the spreading state and the radius of $(A^{\mathbb{Z}}, F)$, respectively. Define $B = A \cup \{s'\}$ where $s' \notin A$. We define the local rule of $(B^{\mathbb{Z}}, G)$ in the following way

$$g(x_1, ..., x_{2r+1}) = \begin{cases} f(x_1, ..., x_{2r+1}) & if \ \ \forall i, x_i \in A \ \ and \ \ \exists x_i \neq s \\ s' & otherwise \end{cases}$$

Note that the new state s' is spreading for $(B^{\mathbb{Z}}, G)$ and that the only block in A^{2r+1} which is mapped to s' is s^{2r+1}. Now, it is clear that $(A^{\mathbb{Z}}, F)$ is nilpotent if and only if $(B^{\mathbb{Z}}, G)$ is nilpotent. Assume that $(A^{\mathbb{Z}}, F)$ is not nilpotent. By compactness, it is possible to prove that there exists a configuration $x \in A^{\mathbb{Z}}$ such that $\forall i \in \mathbb{N}, \forall j \in \mathbb{Z}, F^i(x)_j \neq s$. Define the configuration $y \in B^{\mathbb{Z}}$ in the following way: $y_{(-\infty,-1]} = x_{(-\infty,-1]}, y_{[1,\infty)} = x_{[1,\infty)}$ and $y_0 = s'$. We have that $F^{-1}(y) = \emptyset, \omega(y) = \{...s's's'...\}$ and $\forall i \in \mathbb{N}, \forall j \in \mathbb{Z}, F^i(y)_j \neq s$. For $n \in \mathbb{N}$ consider $z \in F^{-n}(F^n(y))$. Since s is spreading in $A^{\mathbb{Z}}$ and since s^{2r+1} is the unique block in A^{2r+1} which is mapped to s', the only possibility is that $z_0 = s'$. Moreover it is easy to check that $\forall j \in \mathbb{Z} \setminus \{0\}, s' \neq z_j \neq s$ and $F^{-1}(z) = \emptyset$. Then $\forall n \in \mathbb{N}, F^n(y) \notin \Omega_G$ which implies that $(B^{\mathbb{Z}}, G)$ is unstable. ∎

Note that, by Proposition 2.6 and Proposition 2.7, if a property $\mathcal{P}$ is a property of all stable CA then $\mathcal{P}$ is undecidable.

Theorem 3.4. *Every nonempty stable property of limit set dynamics is undecidable.*

Proof. The proof is by reduction from nilpotency. Assume that $\mathcal{P}$ is some nonempty stable property of limit set dynamics. Let $\#(A^{\mathbb{Z}}, F) \in \mathcal{P}$ and let $(B^{\mathbb{Z}}, G)$ be a cellular automaton with a spreading state $s \in B$. By Lemma 3.3, we can assume that $(B^{\mathbb{Z}}, G)$ is stable if and only if it is nilpotent.

We show how to build a new cellular automaton $(C^{\mathbb{Z}}, H)$ such that $\#(C^{\mathbb{Z}}, H) \in \mathcal{P}$ if and only if $(B^{\mathbb{Z}}, G)$ is nilpotent. We can build $(C^{\mathbb{Z}}, H)$ by simply taking the product of $(A^{\mathbb{Z}}, F)$ with $(B^{\mathbb{Z}}, G)$. In detail, consider the product cellular automaton $(A^{\mathbb{Z}} \times B^{\mathbb{Z}}, F \times G)$. To obtain $(C^{\mathbb{Z}}, H)$ it is sufficient to recode the alphabet of $A^{\mathbb{Z}} \times B^{\mathbb{Z}}$ in the following way

$$\forall a \in A, \forall b \in B, (a, b) = \begin{cases} a & if \ b = s \\ a_b & otherwise \end{cases}$$

Since there is a 1-to-1 mapping between $C^{\mathbb{Z}}$ and $A^{\mathbb{Z}} \times B^{\mathbb{Z}}$, the local rule of H on $C^{\mathbb{Z}}$ is naturally induced by the local rule of $F \times G$ on $A^{\mathbb{Z}} \times B^{\mathbb{Z}}$.

Now, it is not difficult to see that $\Omega_F = \Omega_H, F|\Omega_F = H|\Omega_H$ if and only if $(B^{\mathbb{Z}}, G)$ is nilpotent and in this case $\#(C^{\mathbb{Z}}, H) \in \mathcal{P}$. On the contrary $(B^{\mathbb{Z}}, G)$ is unstable then $(C^{\mathbb{Z}}, H)$ is also unstable and $\#(C^{\mathbb{Z}}, H) \notin \mathcal{P}$. ∎

The construction of Theorem 3.4 can be used also for the unique subshift attractor case. Also in this case note that, by Proposition 2.5, if a property $\mathcal{P}$ is a property of all CA with a unique subshift attractor then $\mathcal{P}$ is undecidable.

Theorem 3.5. *Every nonempty unique subshift attractor property of limit set dynamics is undecidable.*

Proof. The proof is by reduction from nilpotency. Let $\mathcal{P}$ be some nonempty unique subshift attractor property of limit set dynamics. Let $\#(A^{\mathbb{Z}}, F) \in \mathcal{P}$ and let $(B^{\mathbb{Z}}, G)$ be a cellular automaton with a spreading state $s \in B$. By using the construction of Theorem 3.4, we can build a cellular automaton $(C^{\mathbb{Z}}, H)$ by taking the product of $(A^{\mathbb{Z}}, F)$ with $(B^{\mathbb{Z}}, G)$. We show that $\#(C^{\mathbb{Z}}, H) \in \mathcal{P}$ if and only if $(B^{\mathbb{Z}}, G)$ is nilpotent. As shown in Theorem 3.4, we have that $\Omega_F = \Omega_H$ and $F|\Omega_F = H|\Omega_H$ if and only if $(B^{\mathbb{Z}}, G)$ is nilpotent. Moreover, by construction, $A \subset C$ and the clopen set $U = \{[a] \mid a \in A\}$ is H-invariant, spreading and $\omega_H(U) = \Omega_F$. Then if $(B^{\mathbb{Z}}, G)$ is nilpotent we have that $\omega_H(U) = \Omega_F = \Omega_H$ and $\#(C^{\mathbb{Z}}, H) \in \mathcal{P}$. Otherwise $\omega_H(U) = \Omega_F \neq \Omega_H$ and $(C^{\mathbb{Z}}, H)$ has two distinct subshift attractors, Ω_F and Ω_H, then $\#(C^{\mathbb{Z}}, H) \notin \mathcal{P}$. ∎

To conclude we show some properties of limit set dynamics which are undecidable. We need the following proposition.

Theorem 3.6. *Let $(A^{\mathbb{Z}}, F)$ be a cellular automaton. If $F : \Omega_F \to \Omega_F$ is closing then Ω_F is a mixing SFT.*

Proof. Since F is closing on Ω_F, $\exists n > 0$ such that $\forall x, y \in \Omega_F, \forall i \in \mathbb{Z}$ if $x_{[i,i+n)} = y_{[i,i+n)}$ and $F(x)_{[i,i+2n]} = F(y)_{[i,i+2n]}$ then $x_{i+n} = y_{i+n}$. Consider the subshift $S = \{(x, y) \mid F(x) = y\} \subseteq \Omega_F \times \Omega_F$. Let $m = \max\{n, r\}$ where r is the radius of $(A^{\mathbb{Z}}, F)$. Let S_{2m+1} be the SFT approximation of order $2m + 1$ of S. Consider the two projections of S_{2m+1}:

- $S'_{2m+1} = \{x \mid \exists (x, y) \in S_{2m+1}\}$
- $S''_{2m+1} = \{y \mid \exists (x, y) \in S_{2m+1}\}$

Since $m \geq r$, we have $F(S'_{2m+1}) = S''_{2m+1}$ and $\Omega_F \subseteq S'_{2m+1}$. We show that S'_{2m+1} is a SFT and that F restricted to S'_{2m+1} is closing. Since $F(S'_{2m+1}) = S''_{2m+1}$, it follows that (S'_{2m+1}, σ) is conjugated to (S_{2m+1}, σ) then S'_{2m+1} is a SFT. Assume for absurd that there are two sequences $x, y \in S'_{2m+1}$ such that $x_n \neq y_n$, $x_{(-\infty,n)} = y_{(-\infty,n)}$ and $F(x) = F(y)$. Then, since $m \geq n$ and F is closing on Ω_F it follows that must be $x_n = y_n$ contradicting the assumption.

Let k be the order of S'_{2m+1} and let $t \in \mathbb{N}$ such that $2t + 1 \geq k$. By Proposition 1.4, there exists an F-invariant clopen set $U \subseteq \mathcal{B}_{2^{-t}}(\Omega_F)$ such that $\omega(U) = \Omega_F$. Moreover, by Proposition 1.8, U contains a mixing SFT Σ such that $F(\Sigma) \subseteq \Sigma$ and $\omega(\Sigma) = \Omega_F$. Moreover, since $2t + 1$ is larger than the order of S'_{2m+1}, we have also $\Omega_F \subseteq \Sigma \subseteq S'_{2m+1}$. Now, since F is closing on S'_{2n+1}, it follows that F must be closing on Σ. Then since Σ is mixing, $F(\Sigma) \subseteq \Sigma$ and F is closing on Σ it follows that $F(\Sigma) = \Sigma$ which implies that $\Sigma \subseteq \Omega_F$ and then $\Omega_F = \Sigma$. ∎

From Theorem 3.4 and Theorem 3.5 we can easily derive the following corollary.

Corollary 3.7. *There is no algorithm that, given $(A^{\mathbb{Z}}, F)$, can decide if*

1. *$F : \Omega_F \to \Omega_F$ is transitive.*
2. *$F : \Omega_F \to \Omega_F$ is closing,*
3. *$F : \Omega_F \to \Omega_F$ is injective,*
4. *$F : \Omega_F \to \Omega_F$ is the identity map,*
5. *$F : \Omega_F \to \Omega_F$ is expansive,*
6. *$F : \Omega_F \to \Omega_F$ is positively expansive,*

Proof. By Theorem 3.4 and Theorem 3.5 it is sufficient to show that properties $1, .., 6$ imply that $(A^{\mathbb{Z}}, F)$ is stable or that it has a unique subshift attractor.

1. If $F : \Omega_F \to \Omega_F$ is transitive then Ω_F is the unique attractor of $(A^{\mathbb{Z}}, F)$ and, in particular, it is the unique subshift attractor.
2. If $F : \Omega_F \to \Omega_F$ is closing then, by Theorem 3.6, Ω_F is a mixing SFT then $(A^{\mathbb{Z}}, F)$ must be stable.
3. If $F : \Omega_F \to \Omega_F$ is injective then, since F is surjective on Ω_F, it must be invertible and then closing.
4. If $F : \Omega_F \to \Omega_F$ is the identity map then F must be injective on Ω_F.
5. If $F : \Omega_F \to \Omega_F$ is expansive then F must be injective on Ω_F and then closing and transitive.
6. If $F : \Omega_F \to \Omega_F$ is positively expansive then F must be closing on Ω_F and transitive.

4. Concluding remarks

In this paper we proved that any property of limit set dynamics is undecidable, if it implies stability or the existence of a unique subshift attractor. As examples of properties which imply stability we have closing (which implies that the limit set is a mixing SFT), injectivity, expansivity, positively expansiveness and identity (all of which imply closing). As examples of properties which imply the existence of a unique subshift attractor we have transitivity, expansivity and positively expansiveness (expansive and positively expansive endomorphisms of mixing SFTs are transitive). From Theorem 3.4 and 3.5 we can conclude that all such properties are undecidable. We remark that, since surjectivity is not a property of limit set dynamics (and it is decidable), if we restrict to only surjective CA then we cannot derive any conclusion from our theorems. In particular we cannot conclude anything about the decidability of transitivity, expansivity and positively expansiveness (it is already known that closing, injectivity and identity are decidable for surjective CA).

Our main undecidability proofs are by reduction from nilpotency. Note that a nilpotent CA is stable and it has a unique subshift attractor. Then (the problem to decide) nilpotency is the easiest problem among all decision problems on the limit set dynamics of stable CA and of CA with a unique subshift attractor.

We conclude the paper by raising a question. It is not clear how stability is related to the existence of a unique subshift attractor. To our knowledge there are no examples of stable CA with two distinct subshift attractors. For a wide class of stable CA it is possible to prove that they have a unique subshift attractor (in particular surjective CA, see [FK07]) but the general question is open. If stable CA have a unique subshift attractor then Lemma 3.3 would be useless and we could derive Theorem 3.4 as a corollary of Theorem 3.5.

Question 4.1. *Is there any stable CA with two distinct subshift attractors?*

References

[FK07] E. Formenti, P. Kůrka. Subshift attractors of cellular automata. *Nonlinearity* 20, no. 1:105–117, 2007.

[Kari92] J. Kari. The nilpotency problem of one-dimensional cellular automata. *SIAM J. on Comp.* 21:571–586, 1992.

[Kari94] J. Kari. Rice's theorem for the limit sets of cellular automata. *Theoret. Comput. Sci.* 127, no. 2: 229–254, 1994.

[Hurd90] L.P. Hurd. Recursive cellular automata invariant sets. *Complex Systems 4*, no. 2: 119–129, 1990.

[LM95] D. Lind, B. Marcus. An introduction to symbolic dynamics and coding. *Cambridge University Press*, Cambridge, 1995.

[Maass95] A. Maass. On the sofic limit sets of cellular automata. *Ergodic Theory Dynam. Systems* 15, no. 4: 663–684, 1995.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 349–360
www.stacs-conf.org

SEMI-ONLINE PREEMPTIVE SCHEDULING: ONE ALGORITHM FOR ALL VARIANTS

TOMÁŠ EBENLENDR [1] AND JIŘÍ SGALL [1]

[1] Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic.
E-mail address: {ebik,sgall}@math.cas.cz

Abstract. We present a unified optimal semi-online algorithm for preemptive scheduling on uniformly related machines with the objective to minimize the makespan. This algorithm works for all types of semi-online restrictions, including the ones studied before, like sorted (decreasing) jobs, known sum of processing times, known maximal processing time, their combinations, and so on. Based on the analysis of this algorithm, we derive some global relations between various semi-online restrictions and tight bounds on the approximation ratios for a small number of machines.

1. Introduction

We study online scheduling on *uniformly related machines*, which means that the time needed to process a job with processing time p on a machine with speed s is p/s. *Preemption* is allowed, i.e., each job may be divided into several pieces, which can be assigned to different machines in disjoint time slots. The objective is to minimize the *makespan*, i.e., the length of a schedule. In the *online* problem, jobs arrive one-by-one and we need to assign each incoming job without any knowledge of the jobs that arrive later. When a job arrives, its assignment at all times must be given and we are not allowed to change this assignment later. In other words, the online nature of the problem is given by the ordering of the input sequence and it is not related to possible preemptions and the time in the schedule.

We focus on *semi-online* algorithms. This term encompasses algorithms that are essentially online, but some partial information about the input is given to the scheduler in advance. The main motivation behind this approach is the observation that the classical competitive analysis is too pessimistic compared to practical results, or, in other words, the adversary who may arbitrarily determine the input sequence is too powerful. In practice, the inputs are not completely arbitrary, and it may be reasonable to restrict the set of inputs. In scheduling, numerous semi-online models have been studied; typical examples include (sequences of) jobs with decreasing processing times, jobs with bounded processing times, sequences with known total processing time of jobs and so on. Most of these models can be viewed as online algorithms on a restricted set of input sequences. Restrictions of

Key words and phrases: On-line algorithms, scheduling.

Partially supported by Institutional Research Plan No. AV0Z10190503, by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR) and grant IAA1019401 of GA AV ČR.

this type have been studied also for other online problems; the most prominent example is paging with locality of reference [1].

Our results

We give a semi-online algorithm for preemptive scheduling on uniformly related machines which is optimal for any chosen semi-online restriction, see Section 2. This means not only the cases listed above—the restriction can be given as an *arbitrary set* of sequences that are allowed as inputs. For any semi-online restriction, the algorithm achieves the best possible approximation ratio for any number of machines and any particular combination of machine speeds; it is deterministic, but its approximation ratio matches the best possible approximation ratio of any randomized algorithm. This generalizes and unifies previous results for various special cases of semi-online preemptive scheduling. We find such a general result providing a provably optimal algorithm for many problems quite exceptional not only in the area of scheduling but also in the whole area of online algorithms. Our result also provides a clear separation between the design of the algorithm and the analysis of the optimal approximation ratio. While the algorithm is always the same, analysis of the optimal ratio depends on the studied restrictions. Nevertheless, the general result also provides crucial new insights and methods and thus we can analyze the optimal ratio in cases that have been out of reach with previously known techniques.

For typical semi-online restrictions, we show that the optimal ratio can be computed by linear programs (with machine speeds as parameters). Studying these linear programs allows us to progress in two directions. First, we are able to completely analyze the optimal ratio for particular cases with a small number of machines. Second, we are able to study the relations between the optimal approximation ratios for different semi-online restrictions and give some bounds for a large number of machines.

The exact analysis of special cases for a small number of machines was given in [6, 3, 11] for various restrictions, and in many more cases for non-preemptive scheduling. Typically, these results involve similar but ad hoc algorithms and an extensive case analysis which is tedious to verify, and can be done for two uniformly related machines or for more identical machines (i.e., all speeds are equal). Using our linear programs we can calculate the ratio as a formula in terms of speeds. This is a fairly routine task which can be simplified (but not completely automated) using standard mathematical software. Once the solution is known, verification amounts to checking the given primal and dual solutions for the linear program. We give a few examples of such results in Section 3; typically the verification is quite simple for $m = 3$ or $m = 4$.

Another research direction is to compute, for a given semi-online restriction, the optimal approximation ratio which works for any number of machines and combination of speeds. This task appears to be much harder, and even in the online case we only know that the ratio is between 2.054 and $e \approx 2.718$; the lower bound is shown by a computer-generated hard instance with no clear structure [4]. Only for identical machines, the exact ratio for any number of machines is known (i) for the online case, where it tends to $e/(e-1) \approx 1.58$ [2], and (ii) for non-increasing processing times, where it tends to $(1+\sqrt{3})/2 \approx 1.366$ [13].

We are able to prove certain relations between the approximation ratios for different restrictions. Some basic restrictions form an inclusion chain: The inputs where the first job has the maximal size (which is equivalent to known maximal size) include the inputs with non-increasing processing times, which in turn include the inputs with all jobs of equal size. Typically, the hard instances have non-decreasing processing times. Thus, one

expected result is that the restriction to non-increasing processing times gives the same approximation ratio as when all jobs have equal size, even for any particular combination of speeds. The overall approximation ratio is at most 1.52, see Section 3.3. On the other hand, for known maximal size of a job we have a computer-generated hard instance with approximation ratio 1.88 with $m = 120$.[1] Thus restricting the jobs to be non-increasing helps the algorithm much more than just knowing the maximal size of a job. This is very different from identical machines, where knowing the maximal size is equally powerful as knowing that all the jobs are equal, see [13].

More interestingly, the overall approximation ratio with known sum of processing times is the same as in the online case—even though for a small fixed number of machines knowing the sum provides a significant advantage. This is shown by a padding argument, see Section 3.1. In fact this is true also in presence of any additional restriction that allows scaling input sequences, taking a prefix, and extending the input by small jobs at the end. Thus, for example, the overall approximation ratio with non-increasing jobs and known sum of processing times is at least 1.366, using the bound for identical machines from [13].

Due to the space limit, some technical parts are available only in the extended version at `http://www.math.cas.cz/~sgall/ps/semirel.pdf`

Preliminaries

Let M_i, $i = 1, 2, \ldots, m$, denote the m machines, and let s_i be the speed of M_i. The machines are sorted by decreasing speeds, i.e., $s_1 \geq s_2 \geq \cdots \geq s_m$. We assume that $s_1 > 0$. The vector of speeds is denoted $\mathbf{s}$, the sum of the speeds is $S = \sum_{i=1}^{m} s_i$ and $S_k = \sum_{i=1}^{k} s_i$ is the sum of the k largest speeds. W.l.o.g., we add infinitely many machines of speed zero, i.e., we put $s_i = 0$ for any $i > m$. (Scheduling a job on one of these zero-speed machines means that we do not process the job at the given time at all.) Let $\mathcal{J} = (p_j)_{j=1}^{n}$ denote the input sequence of jobs, where n is the number of jobs and $p_j \geq 0$ is the size, or the processing time, of the jth job. The sum of processing times is denoted $P = P(\mathcal{J}) = \sum_{j=1}^{n} p_j$. Given $\mathcal{J}$ and $i \leq n$, let $\mathcal{J}_{[i]}$ be the prefix of $\mathcal{J}$ obtained by taking the first i jobs.

The time needed to process a job p_j on machine M_i is p_j/s_i; each machine can process at most one job at any time. Preemption is allowed, which means that each job may be divided into several pieces, which can be assigned to different machines, but any two time slots to which a single job is assigned must be disjoint (no parallel processing of a job); there is no additional cost for preemptions. Formally, if t_i denotes the total length of the time intervals when the job p_j is assigned to machine M_i, it is required that $t_1 s_1 + t_2 s_3 + \cdots + t_m s_m = p_j$. (A job may be scheduled in several time slots on the same machine, and there may be times when a partially processed job is not running at all.) In the (semi-)online version of this problem, jobs arrive one-by-one and at that time the algorithm has to give a complete assignment of this job at all times, without the knowledge of the jobs that arrive later. The objective is to find a schedule of all jobs in which the maximal completion time (the makespan) is minimized.

For an algorithm A, let $C^{\mathsf{A}}_{\max}[\mathcal{J}]$ be the makespan of the schedule of $\mathcal{J}$ produced by A. By $C^{*}_{\max}[\mathcal{J}]$ we denote the makespan of the optimal offline schedule of $\mathcal{J}$. A (randomized) algorithm A is an R-approximation if for every input $\mathcal{J}$, the (expected) makespan is at most R times the optimal makespan, i.e., $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}]] \leq R \cdot C^{*}_{\max}[\mathcal{J}]$.

The optimal makespan can be computed as $C^{*}_{\max}[\mathcal{J}] = \max\{P/S, \max_{k=1}^{m-1}\{P_k/S_k\}\}$, where P_k denotes the sum of the k largest processing times in $\mathcal{J}$ and S_k is the sum of the k

[1] See the Maple output at `http://www.math.cas.cz/~sgall/ps/semirel-pmax.mpl`

largest speeds. This is a lower bound on the makespan, as the first term gives the minimal time when all the work can be completed using all the machines fully, and similarly the term for k is the minimal time when the work of the k largest jobs can be completed using the k fastest machines fully. The tightness of this bound follows from [10, 8, 5].

Semi-online restrictions and previous work

We define a general semi-online input restriction to be simply a set Ψ of allowed inputs, also called *input sequences*. We call a sequence a *partial input* if it is a prefix of some input sequence; the set of all partial inputs is denoted $\mathrm{pref}(\Psi)$. Thus the partial inputs are exactly the sequences that the algorithm can see at some point. A (randomized) semi-online algorithm A with restriction Ψ is an R-approximation algorithm if $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}]] \leq R \cdot C^*_{\max}[\mathcal{J}]$ for any $\mathcal{J} \in \Psi$. Note that this implies that for any prefix $\mathcal{J}'$ of $\mathcal{J}$, $\mathbb{E}[C^{\mathsf{A}}_{\max}[\mathcal{J}']] \leq R \cdot C^*_{\max}[\mathcal{J}]$.

Below we list some of the restrictions that are studied in the literature, together with the notation that we are going to use, the previous work, and our results.

Known sum of processing times, $\sum p_j = P$. For a given value $\bar{P}$, Ψ contains all sequences with $P = \bar{P}$. We prove that the overall ratio is surprisingly the same as in the general online case, on the other hand we note that for $m = 2$, 1-approximation is possible and we analyze the cases of $m = 3, 4$.

Non-increasing processing times, denoted *decr*. Here Ψ contains all sequences with $p_1 \geq p_2 \geq \cdots \geq p_n$. For $m = 2$, the optimal algorithm for all $\mathbf{s}$ was analyzed in [6] and for identical machines in [13]. We prove that for any $\mathbf{s}$ this case is the same as the case with all jobs equal. We analyze the cases for $m = 2, 3$, and prove some bounds for larger m.

Known optimal makespan, $C^*_{\max} = T$. For a given value $\bar{T}$, Ψ contains all sequences with $C^*_{\max}[\mathcal{J}] = \bar{T}$. A 1-approximation semi-online algorithm is known for any $\mathbf{s}$, see [5].

Known maximal job size, $p_{\max} = p$. For a given value $\bar{p}$, Ψ contains all sequences with $\max p_j = \bar{p}$. This is equivalent to the case when the first job is maximal, as any algorithm for that special case can be used also for the case when the maximal job arrives later. Thus this restriction also includes non-increasing jobs. In [13] it is shown that for identical machines, the approximation ratio is the same as when the jobs are non-increasing. We show that this is not the case for general speeds. This restriction was introduced in [9] for non-preemptive scheduling on 2 identical machines.

Tightly grouped processing times. For given values $\bar{p}$ and α, Ψ contains all sequences with $p_j \in [\bar{p}, \alpha\bar{p}]$ for each j. This restriction was introduced in [9] for non-preemptive scheduling on 2 identical machines. Tight bounds for preemptive scheduling on 2 uniformly related machines were given in [3].

Inexact partial information. In this case, some of the previously considered values (optimal makespan, sum of job sizes, maximal job size) is not known exactly but only up to a certain factor. These variants were studied first in [15] without preemption and then in [11] for preemptive scheduling; both on identical machines.

Online scheduling. Here Ψ contains all sequences. In our (i.e., the authors and Wojtek Jawor) previous work [4], we have designed an optimal online algorithm for all speed vectors. The algorithm and the proof of the main result in this paper generalize that result, using the same techniques, however, some technical issues have to be handled carefully to achieve the full generality of our new result. Online preemptive scheduling was studied first in [2].

The paper [12] is probably the first paper which studied and compared several notions of semi-online algorithms, including known sum of processing times. Some combination

of the previous restrictions were studied in [14] for non-preemptive scheduling on identical machines. We should note that there are also semi-online models that do not fit into our framework at all. For example, the algorithm may get a hint which job is the last one, or it is allowed to store some job(s) in a buffer.

2. The optimal algorithm

The new algorithm is based on the algorithm for online scheduling from [4]. In this section we present the key ideas with emphasis on the issues that need to be handled differently in the more general semi-online setting.

Suppose that we are given a parameter r and we try to develop an r-approximation algorithm. In the online case, we simply make sure that the current job completes by time r times the current optimal makespan. In the semi-online case, if the restriction is not closed under taking a prefix, this would be too pessimistic. It may happen that the current partial input is not in Ψ and we know that any extension in Ψ has much larger optimal makespan (e.g., if the restriction forces that some large jobs will arrive later). In this case we can schedule the current job so that it complete much later than at time r times the current optimal makespan. For this purpose, we define the appropriate quantity to be used instead of the current optimal makespan.

Definition 2.1. For an input restriction Ψ and a partial input $\mathcal{I} \in \mathrm{pref}(\Psi)$, we define the optimal makespan as the infimum over all possible end extensions of $\mathcal{J}$ that satisfy Ψ:

$$C^{*,\Psi}_{\max}[\mathcal{I}] = \inf\{C^{*}_{\max}[\mathcal{J}] \mid \mathcal{J} \in \Psi \ \&\ \mathcal{I} \text{ is a prefix of } \mathcal{J}\}$$

Note that for any input sequence $\mathcal{J} \in \Psi$ we have $C^{*}_{\max}[\mathcal{J}] = C^{*,\Psi}_{\max}[\mathcal{J}]$.

Algorithm RatioStretch

Our algorithm takes as a parameter a number r which is the desired approximation ratio. Later we show that, for the right choice of r, our algorithm is optimal. Given r, we want to schedule each incoming job so that it completes at time $r \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$. By the definition of $C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$, any schedule for any possible extension of the current partial input will have makespan at least $C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$, in particular $C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}] \le C^{*}_{\max}[\mathcal{J}]$. Thus, if each job j completes by time $r \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}] \le r \cdot C^{*}_{\max}[\mathcal{J}]$, we have an r-approximation algorithm.

Even when we decide the completion time of a job, there are many ways to schedule it given the flexibility of preemptions. We choose a particular one based on the notion of a *virtual machine* from [5, 4]. We define the ith *virtual machine*, denoted V_i, so that at each time τ it contains the ith fastest machine among those real machines M_1, M_2, ..., M_m that are idle at time τ. Due to preemptions, a virtual machine can be thought and used as a single machine with changing speed. When we schedule (a part of) a job on a virtual machine during some interval, we actually schedule it on the corresponding real machines that are uniquely defined at each time.

Upon arrival of a job j we compute a value T_j defined as $r \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]$. Then we find two adjacent virtual machines V_k and V_{k+1}, and time t_j, such that if we schedule j on V_{k+1} in the time interval $(0, t_j]$ and on V_k from t_j on, then j finishes exactly at time T_j.

We need to show that we can always find such machines V_k and V_{k+1}. Since we have added the machines of speed 0, it only remains to prove that each job can fit on V_1. This is true for the appropriate value of r.

Before we sketch the proof, we make a few remarks concerning efficiency and uniformity of the algorithm. The only parts of the algorithm that depend on the semi-online restriction are (i) the computation of the optimal approximation ratio and (ii) the computation of $C^{*,\Psi}_{\max}[\mathcal{J}]$. The rest of the algorithm is independent of the restriction and very efficient. Similarly to the online algorithms, for semi-online algorithms we generally do not require the computation to be polynomial time. For a general restriction the optimal algorithm cannot be efficient. (If the set of input sequences is, e.g., not recursive, then it may be algorithmically undecidable how much time we have even for scheduling the first job. Besides, there are more possible restrictions than algorithms.) Nevertheless, the algorithm is efficient for many natural restrictions. Computing $C^{*,\Psi}_{\max}[\mathcal{J}]$ is usually simple. If the restriction is closed under taking prefixes, then it is equal to $C^{*}_{\max}[\mathcal{J}]$. In other cases it is easy to see which extension has the smallest makespan. Computing the optimal approximation ratio is more difficult, but in Section 3 it is shown that in many natural cases it reduces to linear programming. Alternatively, we can use any upper bound on the approximation ratio and give to the algorithm as a parameter.

Optimality of Algorithm RatioStretch

Our goal is to show that Algorithm RatioStretch works whenever the parameter r is at least the optimal approximation ratio for the given Ψ and $\mathbf{s}$. We actually prove the converse: Whenever for some input $\mathcal{J}$ Algorithm RatioStretch with the parameter r fails, we prove that there is no r-approximation algorithm.

This is based on a generalization of a lemma from [7] which provides the optimal lower bounds for online algorithms, as shown in [4]. The key observation in its proof is this: On an input $\mathcal{J}$, if the adversary stops the input sequence at the ith job from the end, any r-competitive online algorithm must complete by time r times the current optimal makespan, and after this time, in the schedule of $\mathcal{J}$, only $i-1$ machines can be used. This bounds the total work of all the jobs in terms of r and optimal makespans of the prefixes, and thus gives a lower bound on r. To generalize to an arbitrary restriction Ψ, we need to deal with two issues.

First, the adversary cannot stop the input if the current partial input is not in Ψ. Instead, the sequence then must continue so that its optimal makespan is the current $C^{*,\Psi}_{\max}$ (or its approximation). Consequently, the bound obtained uses $C^{*,\Psi}_{\max}$ in place of previous $C^{*}_{\max}$, which possibly decreases the obtained bound.

Second, for a general semi-online restriction, using the last m prefixes of $\mathcal{J}$ may not give the best possible lower bound. E.g., the restriction may force that some job is tiny, and thus using the prefix ending at this job is useless; in general, we also cannot remove such a job from the input sequence. To get a stronger lower bound, we choose a subsequence of important jobs from $\mathcal{J}$ and bound their total work in terms of values $C^{*,\Psi}_{\max}$ of the prefixes of the original sequence $\mathcal{J}$.

Lemma 2.2. *Let* A *be any randomized R-approximation semi-online algorithm for preemptive scheduling on m machines with an input restriction Ψ. Then for any partial input $\mathcal{J} \in \mathrm{pref}(\Psi)$, for any k, and for any subsequence of jobs $1 \le j_1 < j_2 < \cdots < j_k \le n$ we have*

$$\sum_{i=1}^{k} p_{j_i} \quad \le \quad R \cdot \sum_{i=1}^{k} s_{k+1-i} C^{*,\Psi}_{\max}[\mathcal{J}_{[j_i]}].$$

Let r^Ψ be the largest lower bound on the approximation ratio obtained by Lemma 2.2:

Definition 2.3. For any vector of speeds $\mathbf{s}$ and any partial input $\mathcal{J} \in \mathrm{pref}(\Psi)$,

$$r^\Psi(\mathbf{s}, \mathcal{J}) = \sup_{1 \le j_1 < j_2 < \cdots < j_k \le n} \frac{\sum_{i=1}^{k} p_{j_i}}{\sum_{i=1}^{k} s_{k+1-i} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j_i]}]}.$$

For any $\mathbf{s}$, let $r^\Psi(\mathbf{s}) = \sup_{\mathcal{J} \in \mathrm{pref}(\Psi)} r^\Psi(\mathbf{s}, \mathcal{J})$. Finally, let $r^\Psi = \sup_{\mathbf{s}} r^\Psi(\mathbf{s})$.

With these definitions and Lemma 2.2, we can prove the following main theorem. If Algorithm RatioStretch cannot schedule the incoming job, we choose a subsequence including the jobs scheduled so far on the first virtual machine and the incoming job. We use Lemma 2.2 with this subsequence to argue that that no (randomized) algorithm can have the same approximation ratio.

Theorem 2.4. *For any restriction Ψ and vector of speeds $\mathbf{s}$, Algorithm* RatioStretch *with a parameter $r \ge r^\Psi(\mathbf{s})$ is an r-approximation algorithm for semi-online preemptive scheduling on m uniformly related machines. In particular, $r^\Psi(\mathbf{s})$ (resp. r^Ψ) is the optimal approximation ratio for semi-online algorithms for Ψ with speeds $\mathbf{s}$ (resp. with arbitrary speeds).*

3. Reductions and linear programs

We have a formula for $r^\Psi(\mathbf{s})$ which gives the desired approximation ratio for any speeds and Ψ as a supremum over a bound for all partial inputs and all their subsequences. It is not obvious how to turn this into an efficient algorithm. Now we develop a general methodology how to compute the ratio using linear programs and apply it to a few cases.

We observed that for a general restriction it may be necessary to use an arbitrary subsequence in Definition 2.3. However, for many restrictions it is sufficient to use the whole sequence, similarly as for online scheduling. Usual restrictions are essentially of two kinds. The first type are the restrictions that put conditions on individual jobs or their order. These restrictions are closed under taking subsequences (not only prefixes), i.e., any subsequence of an input sequence is also in Ψ. The second type are the restrictions where some global information is given in advance, like $\sum p_j = P$ or $C^*_{\max} = T$. These are not closed under taking subsequences, but are closed under permuting the input sequence.

We define a large class of restrictions that includes both types of restrictions discussed above as well as their combinations; in particular it includes all the restrictions listed and studied here. The definition below implies that any subsequence of any input sequence is a prefix of another input. Thus, the sets of all the subsequences and all the prefixes of Ψ coincide, and Definition 2.3 simplifies using the monotonicity condition in the definition.

Definition 3.1. An input restriction Ψ is *proper* if for any $\mathcal{J} \in \Psi$ and any subsequence $\mathcal{I}$ of $\mathcal{J}$, we have $\mathcal{I} \in \mathrm{pref}(\Psi)$ and furthermore $C^{*,\Psi}_{\max}[\mathcal{I}] \le C^{*,\Psi}_{\max}[\mathcal{J}]$.

Definition 3.2. Let Ψ be a proper semi-online restriction and $\mathcal{J} \in \mathrm{pref}(\Psi)$ a partial input. We define

$$\bar{r}^\Psi(\mathbf{s}, \mathcal{J}) = \frac{\sum_{j=1}^{n} p_j}{\sum_{j=1}^{n} s_{n+1-j} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]}.$$

From now on, we focus on proper restrictions. It may happen that $r^\Psi(\mathbf{s},\mathcal{J}) > \bar{r}^\Psi(\mathbf{s},\mathcal{J})$. By Definitions 2.3 and 2.3 we may take a subsequence of jobs $\mathcal{I} = (p_{j_i})_{i=1}^k$ that achieves the value of $\bar{r}^\Psi(\mathbf{s},\mathcal{I}) \geq r^\Psi(\mathbf{s},\mathcal{J}) - \varepsilon$ for any $\varepsilon > 0$. By the definition of a proper restriction, $\mathcal{I} \in \mathrm{pref}(\Psi)$. Taking the supremum over all partial inputs, we obtain the following simpler formula for the optimal approximation ratio.

Observation 3.3. For any proper restriction Ψ, $r^\Psi(\mathbf{s}) = \sup_{\mathcal{J}\in\mathrm{pref}(\Psi)} \bar{r}^\Psi(\mathbf{s},\mathcal{J})$.

Our strategy is to reduce the number of sequences $\mathcal{J}$ that need to be taken into account. Typically, we show that the sequences must be sorted. Then we know which jobs are the biggest ones and we can express the optimal makespans for prefixes by linear constraints in job sizes. Maximizing the expression for $\bar{r}^\Psi(\mathbf{s})$, which gives the desired bound, is then reduced to solving one or several linear programs. The following observation helps us to limit the set of relevant sequences.

Observation 3.4. Let Ψ be arbitrary proper restriction, let $\mathbf{s}$ be arbitrary speed vector, and let $\mathcal{J},\mathcal{J}' \in \mathrm{pref}(\Psi),$. be two partial inputs with n jobs. Suppose that for some $b > 0$:

$$\begin{aligned} \sum_{j=1}^n p'_j &= b \cdot \sum_{j=1}^n p_j, \quad \text{and} \\ (\forall i = 1,\ldots,n) \quad C^{*,\Psi}_{\max}[\mathcal{J}'_{[i]}] &\leq b \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]. \end{aligned}$$

Then $\bar{r}(\mathbf{s},\mathcal{J}') \geq \bar{r}(\mathbf{s},\mathcal{J})$.

The observation follows immediately from the definition of $\bar{r}^\Psi(\mathbf{s},\mathcal{J})$.

Whenever (i) Ψ is closed under permutations of the sequence and (ii) increasing the size of the last job of a partial input cannot decrease $C^{*,\Psi}_{\max}$, the observation implies that it is sufficient to consider sequences of non-decreasing jobs: If $\mathcal{J}$ contains two jobs with $p_k < p_{k+1}$, swapping them can only increase $C^{*,\Psi}_{\max}[\mathcal{J}_{[k]}]$ and any other $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]$ remains unchanged; thus the observation applies with $b = 1$.

3.1. Known sum of processing times, $\sum p_j = P$

Here we are given a value $\bar{P}$ and Ψ contains all $\mathcal{J}$ with $P = \bar{P}$. It can be easily verified that $C^{*,\Psi}_{\max}[\mathcal{J}] = \max\{C^*_{\max}[\mathcal{J}], \bar{P}/S\}$ for any $\mathcal{J}$ with $P \leq \bar{P}$.

Since we can permute the jobs and increasing the size of the last job does not decrease $C^{*,\Psi}_{\max}$, Observation 3.4 implies that we can restrict ourselves to non-decreasing sequences $\mathcal{J}$. Furthermore, we may assume that $P = \bar{P}$: We know that $P \leq \bar{P}$, as otherwise $\mathcal{J}$ is not a partial input. If $P < \bar{P}$, we scale up $\mathcal{J}$ to $\mathcal{J}'$ by multiplying all the sizes by $b = P'/P$. Observation 3.4 then applies, as each $C^{*,\Psi}_{\max}[\mathcal{J}'_{[i]}] = \max\{C^*_{\max}[\mathcal{J}'_{[i]}], \bar{P}/S\}$ increases by at most the scaling factor b. Finally, we observe that we can restrict ourselves to sequences $\mathcal{J}$ with less than m jobs. If $n \geq m$, we use the fact that $C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}] \geq \bar{P}/S$ for any i and obtain $\bar{r}^\Psi(\mathbf{s},\mathcal{J}) = P/(\sum_{i=1}^n s_{n+1-i} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[i]}]) \leq P/(\sum_{i=1}^n s_{n+1-i} \cdot \bar{P}/S) = 1$, using $n \geq m$ in the last step.

Summarizing, we can assume that $\mathcal{J}$ is a non-decreasing sequence of $n < m$ jobs with $P = \bar{P}$. (Note that this does not mean that the adversary uses fewer jobs than machines, as he may need to release some small jobs at the end of the prefix sequence, to extend it to a sequence in Ψ.) To obtain the worst case bound, we compute $m - 1$ linear programs,

one for each value of n, and take the maximum of their solutions. The linear program for a given P, $\mathbf{s}$, and n has variables q_i for job sizes and O_i for optimal makespans of the prefixes:

$$\begin{aligned}
\textbf{minimize} \quad & r^{-1} = \frac{s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1}{\bar{P}} \\
\textbf{subject to} \quad & \\
q_1 + \cdots + q_n &= \bar{P} \\
\bar{P} &\le (s_1 + s_2 + \cdots + s_m) O_k && \text{for } k = 1, \ldots, n \\
q_j + q_{j+1} + \cdots + q_k &\le (s_1 + s_2 + \cdots + s_{k-j+1}) O_k && \text{for } 1 \le j \le k \le n \\
0 \le q_j &\le q_{j+1} && \text{for } j = 1, \cdots, n-1
\end{aligned}$$

If we fix the input sequence, i.e., the values of q_i, then the smallest objective is achieved for O_k as small as possible which is exactly the value of the optimal makespan, by the constraints involving O_k. Thus the linear program computes correctly the value $1/r^{\sum p_j = P}(\mathbf{s})$. We can also see that the linear program scales and the optimum does not depend on $\bar{P}$.

We now examine the special cases of $m = 2, 3$. The linear program is trivial for $n = 1$, and we conclude that for $m = 2$ the approximation ratio is equal to 1, i.e., RatioStretch always produces an optimal schedule. We can see this also intuitively: The algorithm starts scheduling the incoming jobs in the interval $[0, T_1)$ where $T_1 \ge \bar{P}/S$. Consider the first time when a job is scheduled at the first real machine M_1. It is always possible to schedule this job at the empty machine M_1 so that it completes before the current optimal makespan. Furthermore, after M_1 is used the first time, the algorithm guarantees that in the interval $[0, T_1)$ there is only one real machine idle at any time. This in turn implies that the remaining jobs can be completed by time T_1, as the total size of all jobs is $\bar{P} \le S \cdot T_1$.

For $m = 3$, it remains to solve the linear program for $n = 2$. The resulting ratio is:

$$r^{\sum p_j = P}(s_1, s_2, s_3) = \begin{cases} \dfrac{s_1(s_1 + s_2)}{s_1^2 + s_2^2} & \text{for } s_1^2 \le s_2(s_2 + s_3) \\ 1 + \dfrac{s_2 s_3}{s_1(s_1+s_2+s_3) + s_2(s_1+s_2)} & \text{for } s_1^2 \ge s_2(s_2 + s_3) \end{cases}$$

The overall worst case ratio for three machines is $\frac{2+\sqrt{2}}{3} \approx 1.138$ for $s_1 = \sqrt{2}, s_2 = s_3 = 1$.

Padding. We prove a theorem that shows that knowing the total size of jobs does not improve the overall approximation ratio. This may sound surprising, as for two machines, knowing the sum allows to generate an optimal schedule, and also for three machines the improvement is significant. The same result holds also in presence of an additional restriction with suitable properties. Among the restrictions that we consider, the requirements are satisfied for non-increasing jobs, known maximal job size, or the online case. By "$\Psi, \sum p_j = P$" we denote the intersection of the two restrictions, i.e., the set of all sequences $(p_j)_{j=1}^n \in \Psi$ such that $\sum_{i=1}^n p_j = \bar{P}$ for a given value of $\bar{P}$

We say that Ψ allows scaling if for any $\mathcal{J} \in \Psi$ and $b > 0$, the modified sequence $\mathcal{J}' = (bp_j)_{j=1}^n$ satisfies $\mathcal{J}' \in \Psi$. We say that Ψ allows padding if for any $\mathcal{J} \in \Psi$, there exists $\varepsilon_0 > 0$ such that any sequence $\mathcal{J}'$ created by extending $\mathcal{J}$ by an arbitrary number of equal jobs of size $\varepsilon < \varepsilon_0$ at the end satisfies $\mathcal{J}' \in \Psi$.

Theorem 3.5. *Suppose that Ψ is proper, allows scaling, padding, and is closed under taking prefixes. Let $\mathcal{J} \in \Psi$ and let $\mathbf{s}$ be arbitrary. Then for any $\delta > 0$ there exists $\mathcal{J}'$ and $\mathbf{s}'$ such that $\bar{r}^{\Psi, \sum p_j = P}(\mathbf{s}', \mathcal{J}') \ge \bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})/(1 + \delta)$. Consequently, $r^{\Psi, \sum p_j = P} = r^{\Psi}$.*

Proof. We fix $\mathbf{s}$, $\mathcal{J}$, and $\bar{P}$ given to the algorithm with the restriction $\sum p_j = P$. We proceed towards constructing the appropriate $\mathbf{s}'$ and $\mathcal{J}'$.

Since Ψ allows scaling, the value $C^{*,\Psi}_{\max}[\mathcal{J}]$ is multiplied by b when $\mathcal{J}$ is scaled by b. Consequently, the value of $\bar{r}^{\Psi}(\mathbf{s},\mathcal{J})$ does not change when $\mathcal{J}$ is scaled. Let $\mathcal{J}' = (p'_j)_{j=1}^n$ be the sequence $\mathcal{J}$ scaled so that $\sum_{j=1}^n p'_j = \bar{P}$. Then $\bar{r}^{\Psi}(\mathbf{s},\mathcal{J}') = \bar{r}^{\Psi}(\mathbf{s},\mathcal{J})$.

Choose a small $\sigma > 0$ so that $\sigma < s_m$ and $\sigma < \delta S/n$. Let $O_1 = p'_1/s_1$, i.e., the optimal makespan after the first job. Let $\mathbf{s}'$ be the sequence of speeds starting with $\mathbf{s}$ and continuing with $n + \bar{P}/(O_1\sigma)$ of values σ. The first condition on σ guarantees that $\mathbf{s}'$ is monotone and thus a valid sequence of speeds. The second condition guarantees that the added machines are sufficiently slow, so that for any sequence of at most n jobs, in particular for the prefixes of $\mathcal{J}'$, the makespan decreases by at most the factor of $(1+\delta)$. Since Ψ is closed under taking prefixes, $C^{*,\Psi}_{\max}$ equals $C^{*}_{\max}$ for any sequence. Thus we conclude that $\bar{r}^{\Psi}(\mathbf{s}',\mathcal{J}') \geq \bar{r}^{\Psi}(\mathbf{s},\mathcal{J}')/(1+\delta)$.

Finally, we have added sufficiently many new machines so that for any sequence of at most n jobs, the empty new machines can accommodate total work of $\bar{P}$ without exceeding makespan O_1. This implies that for all prefixes of $\mathcal{J}'$, $C^{*,\Psi,\sum p_j=P}_{\max}[\mathcal{J}'_{[i]}] = C^{*,\Psi}_{\max}[\mathcal{J}'_{[i]}]$; thus $\bar{r}^{\Psi,\sum p_j=P}(\mathbf{s}',\mathcal{J}') = \bar{r}^{\Psi}(\mathbf{s}',\mathcal{J}') \geq \bar{r}^{\Psi}(\mathbf{s},\mathcal{J}')/(1+\delta) \geq \bar{r}^{\Psi}(\mathbf{s},\mathcal{J})/(1+\delta)$. ∎

3.2. Known maximal processing time, $p_{\max} = p$

Here we are given $\bar{p}$, the maximal size of a job. As noted before, any algorithm that works with the first job being the maximal one can be easily changed to a general algorithm for this restriction. First it virtually schedules the maximal job and then it compares the size of each job to $\bar{p}$. If it is equal for the first time, it schedules the job to the time slot(s) it reserved by virtual scheduling at the beginning. Other jobs are scheduled in the same way in both algorithms. Thus we can work with the equivalent restriction containing all the sequences where the first job is maximal. Then $C^{*,\Psi}_{\max}[\mathcal{J}] = C^{*}_{\max}[\mathcal{J}]$ for any partial input. By Observation 3.4, the other jobs can be reordered as in the previous case, and we can maximize only over sequences with non-decreasing job sizes from the second job on.

In this case we are able to use a single linear program to cover input sequences of an arbitrary length. The variables are: p for the length of the first job, q_1 for the total length of jobs $p_2, \ldots, p_{n-m+1}$, and $q_2, \ldots, q_m$ for the jobs $p_{n-m+2}, \ldots, p_n$. For sequences with $n < m$, we set $q_1 = q_2 = \cdots q_{n-m} = 0$. Consider the following non-linear program:

$$\textbf{maximize} \quad r = \frac{P}{s_1 O_m + s_2 O_{m-1} + \cdots + s_m O_1}$$

subject to

$$\begin{aligned}
p + q_1 + \cdots + q_m &= P \\
p + q_1 + \cdots + q_k &\leq (s_1 + s_2 + \cdots + s_m) O_k && \text{for } k = 1, \ldots, m \\
p + q_{j+1} + q_{j+2} + \cdots + q_k &\leq (s_1 + s_2 + \cdots + s_{k-j+1}) O_k && \text{for } 1 \leq j \leq k \leq m \\
0 \leq q_j &\leq q_{j+1} \leq p && \text{for } j = 2, \cdots, m-1 \\
0 &\leq q_1
\end{aligned}$$

If we fix the values of q_i, then the largest objective is achieved for O_k as small as possible. By the constraints involving O_k, this is exactly the value of the optimal makespan for a sequence where q_1 represents a prefix of a sequence of jobs smaller than q_2. Thus the program computes correctly the value $r^{\sum p_j=P}(\mathbf{s})$. The program scales, thus we can scale any feasible solution so that the denominator of the objective function is a given constant. Thus we get an equivalent linear program after adding the constraint $1 = s_1 O_m + s_2 O_{m-1} + \cdots + s_m O_1$.

Small number of machines. For two machines we get the approximation ratio

$$r^{p_{\max}=p}(s_1, s_2) = 1 + \frac{s_1 s_2}{(s_1+s_2)^2 + s_1^2}$$

The maximum is 1.2 for $s_1 = s_2$. For three machines we get

$$r^{p_{\max}=p}(s_1, s_2, s_3) = \begin{cases} 1 + \dfrac{s_1(s_2+s_3)}{S^2 + s_1^2} & \text{for } s_1 s_2 \geq s_3 S \\ 1 + \dfrac{s_1 s_2 + 2 s_1 s_3}{S^2 + 2s_1^2 + s_1 s_2} & \text{for } s_1 s_2 \leq s_3 S \end{cases}$$

This is maximized at $s_1 = 2, s_2 = s_3 = \sqrt{3}$ which gives the ratio $(8 + 12\sqrt{3})/23 \approx 1.252$.

3.3. Non-increasing processing times, *decr*

We are also interested in sequences of non-increasing jobs, as this is one of the most studied restrictions. Now Ψ contains sequences which have $p_j \geq p_{j+1}$ for all j. We cannot swap jobs, however, we can take two adjacent jobs j and $j+1$ and replace both of them by jobs of the average size $(p_j + p_{j+1})/2$. By Observation 3.4, the approximation ratio does not decrease. Similarly, we can replace longer segment of jobs with only two distinct sizes by the same number of jobs of the average size. Repeating this process, we can conclude that for the worst case for a given set of speeds it is sufficient to consider sequences where all jobs have equal size. By scaling, the actual size of jobs does not matter, we only need to determine the length of the sequence which gives the highest ratio.

Let us denote $\hat{r}_n(\mathbf{s}) = \bar{r}^{decr}(\mathbf{s}, \mathcal{J})$ for a sequence $\mathcal{J}$ with n jobs with $p_j = 1$. For this sequence, $C^{*,\Psi}_{\max}[\mathcal{J}] = C^{*}_{\max}[\mathcal{J}] = n/S_n$. (Recall that $s_i = 0$ for $i > m$ and $S_k = \sum_{i=1}^{k} s_i$.) Using this for the prefixes, we obtain from Observation 3.3 that

$$\hat{r}_n(\mathbf{s}) = n \cdot \left(\sum_{k=1}^{n} \frac{k s_{n-k+1}}{S_k} \right)^{-1} .$$

It can be seen that for any speed vector, the sequence $\hat{r}_n(\mathbf{s})$ decreases with n for $n \geq 2m$. Thus computing the approximation ratio for any given speeds is efficient.

A natural approach to estimate the overall ratio is to find for each n the worst speed vector and the corresponding ratio $\hat{r}_n = \sup_{\mathbf{s}} \hat{r}_n(\mathbf{s})$. Based on numerical experiments, we conjecture that for each n, $\hat{r}_n$ is attained for some $\mathbf{s}$ with $s_1 = s_2 = \cdots = s_{m-1}$. I.e., almost all the speeds are equal. This conjecture would imply that with non-increasing jobs, the optimal overall approximation ratio is the same for the uniformly related machines and for the identical machines, and this is equal to $(1 + \sqrt{3})/2 \approx 1.366$ by [13].

This is related to an intriguing geometric question. Suppose we have numbers x_i, y_i, $i = 1, \ldots, n$ such that $x_i y_i = i$ for all i and both sequences $(x_i)_{i=1}^n$ and $(y_i)_{i=1}^n$ are non-decreasing. Consider the union of rectangles $[0, x_i] \times [0, y_{n+1-i}]$ over all i; this is a staircase-like part of the positive quadrant of the plane. What is the smallest possible area of this union of rectangles? We conjecture that the minimum is attained for some $y_1 = y_2 = \ldots = y_k$ and $x_{k+1} = x_{k+2} = \ldots = x_n$ for some k. This would imply the previous conjecture.

We are not able to determine exactly the values of $\hat{r}_n$, but we can prove certain relations between these values. In particular, for any integers a, n, and n', $r_{an} \geq r_n$ and $r_{n'} \leq \frac{n+1}{n} r_n$. For the first proof, we replace a sequence of speeds from the bound for r_n by a sequence where each speed is repeated a times, and the bound follows by manipulating the formula

for r_n. The second inequality is shown by replacing the speeds for $r_{n'}$ by a shorter sequence where each new speed is a sum of a segment of a speeds in the original sequence, for a suitable a. These relations show that whenever we are able to evaluate some r_n for a fixed n, the optimal overall ratio is at most $\frac{n+1}{n} r_n$.

For $n = 3$, maximizing the function $\hat{r}_n(\mathbf{s})$ can be done by hand and the maximum is $r_3 = 1.2$ for $s_1 = s_2 = 1$, $s_3 = 0$. This yields an overall upper bound of $\hat{r}_n \leq \frac{4}{3} \cdot \frac{6}{5} = 1.6$. By a computer-assisted proof we have shown that $\hat{r}_4 = (\sqrt{7}+1)/3 \approx 1.215$, yielding an overall upper bound of $\hat{r}_n \leq \frac{5}{4}\hat{r}_4 = \frac{5}{12}(\sqrt{7}+1) \approx 1.52$.

Conclusions. Similar methods can be used to analyze other semi-online restrictions, their combinations and inexact versions, or give formulas for the approximation ratios for more machines. This becomes a somewhat mechanical exercise; we have not found any surprising phenomenon in the cases we have examined so far.

It would be interesting, and it seems hard to us but not impossible, to determine the exact overall approximation ratios for the basic restrictions.

References

[1] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. *J. Comput. Systems Sci.*, 50:244–258, 1995.

[2] B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Oper. Res. Lett.*, 18:127–131, 1995.

[3] D. Du. Optimal preemptive semi-online scheduling on two uniform processors. *Inform. Process. Lett.*, 92(5):219–223, 2004.

[4] T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. In *Proc. 13th European Symp. on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Comput. Sci.*, pages 327–339. Springer, 2006.

[5] T. Ebenlendr and J. Sgall. Optimal and online preemptive scheduling on uniformly related machines. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 199–210. Springer, 2004.

[6] L. Epstein and L. M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. *Oper. Res. Lett.*, 30:269–275, 2002.

[7] L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Oper. Res. Lett.*, 26(1):17–22, 2000.

[8] T. F. Gonzales and S. Sahni. Preemptive scheduling of uniform processor systems. *J. ACM*, 25:92–101, 1978.

[9] Y. He and G. Zhang. Semi on-line scheduling on two identical machines. *Computing*, 62(3):179–187, 1999.

[10] E. Horwath, E. C. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *J. ACM*, 24:32–43, 1977.

[11] Y. Jiang and Y. He. Optimal semi-online algorithms for preemptive scheduling problems with inexact partial information. *Theoret. Comput. Sci.*, 44(7-8):571–590, 2007.

[12] H. Kellerer, V. Kotov, M. G. Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Oper. Res. Lett.*, 21:235–242, 1997.

[13] S. Seiden, J. Sgall, and G. J. Woeginger. Semi-online scheduling with decreasing job sizes. *Oper. Res. Lett.*, 27:215–221, 2000.

[14] Z. Tan and Y. He. Semi-on-line problems on two identical machines with combined partial information. *Oper. Res. Lett.*, 30:408–414, 2002.

[15] Z. Tan and Y. He. Semi-online scheduling problems on two identical machines with inexact partial information. *Theoret. Comput. Sci.*, 377(1-3):110–125, 2007.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 361–372
www.stacs-conf.org

IMPROVED APPROXIMATIONS FOR GUARDING 1.5-DIMENSIONAL TERRAINS

KHALED ELBASSIONI [1] AND ERIK KROHN [2] AND DOMAGOJ MATIJEVIĆ [3] AND JULIÁN MESTRE [1] AND DOMAGOJ ŠEVERDIJA [3]

[1] Max-Planck-Institut für Informatik,
Saarbrücken, Germany.
E-mail address: {elbassio,jmestre}@mpi-inf.mpg.de

[2] Department of Computer Science,
University of Iowa, Iowa City, USA.
E-mail address: erik-krohn@uiowa.edu

[3] Department of Mathematics,
J. J. Strossmayer University of Osijek, Croatia.
E-mail address: {domagoj,dseverdi}@mathos.hr

Abstract. We present a 4-approximation algorithm for the problem of placing the fewest guards on a 1.5D terrain so that every point of the terrain is seen by at least one guard. This improves on the currently best approximation factor of 5 (see [14]). Unlike most of the previous techniques, our method is based on rounding the linear programming relaxation of the corresponding covering problem. Besides the simplicity of the analysis, which mainly relies on decomposing the constraint matrix of the LP into totally balanced matrices, our algorithm, unlike previous work, generalizes to the weighted and partial versions of the basic problem.

1. Introduction

In the *1.5D terrain guarding* problem we are given a polygonal region in the plane determined by an x-monotone polygonal chain, and the objective is to find the minimum number of guards to place on the chain such that every point in the polygonal region is guarded. This kind of guarding problems and its generalizations to 3-dimensions are motivated by optimal placement of antennas for communication networks; for more details see [4, 1] and the references therein.

1998 ACM Subject Classification: G.2.1.

Key words and phrases: Covering Problems, Guarding 1.5-Terrains, Approximation Algorithms, Linear Programming, Totally Balanced Matrices.

This paper combines the results of [7] and [19], which were obtained independently: the first gave a 4-approximation for the weighted version of the guarding problem and an extension to partial covering, and the second gave a 4-approximation for the unweighted version.

One can easily see that one point is enough to guard the polygonal region if we are allowed to select guards anywhere in the plane. However, the problem becomes interesting if guards can only be placed on the boundary chain. Under this restriction, two natural versions of the problem arise: in the *continuous* version the guards can be placed anywhere along the chain and all points in the terrain must be guarded, while in the *discrete* version the guards and points to be guarded are arbitrary subsets of the chain.

1.1. Previous Work

Chen et al. [4] claimed that the 1.5D-terrain guarding problem is NP-hard, but a complete proof of the claim was never published [6, 14, 1]. They also gave a linear time algorithm for the *left-guarding* problem, that is, the problem of placing the minimum number of guards on the chain such that each point of the chain is guarded from its left. Based on purely geometric arguments, Ben-Moshe *et al.* [1] gave the first constant-factor approximation algorithm for the 1.5D-terrain guarding problem. Although they did not state the value of the approximation ratio explicitly, it was claimed to be at least 6 in [14]. Clarkson et al. [5] gave constant factor approximation algorithms for a more general class of problems using ϵ-nets and showed that their technique can be used to get a constant approximation for the 1.5D-terrain guarding problem. Most recently, King [14] claimed that the problem can be approximated with a factor of 4, but the analysis turned out to have an error that increases the approximation factor to 5 [13].

1.2. Our results and outline of the paper

The main building block of our algorithms is an LP-rounding algorithm for one-sided guarding: A version of the problem where a guard can see either to the left or to the right. Guided by an optimal fractional solution, we can partition the points into those that should be guarded from the left, and those that should be guarded from right. This turns out to be a very useful information since we can show that the LPs for the left-guarding and right-guarding problems are integral. We prove this by establishing a connection between the guarding problem and totally balanced covering problems that is of independent interest. Altogether, this leads to a factor 2 approximation for one-sided guarding. Then we show how to reduce other variants of the problem to the one-sided case by incurring an extra multiplicative factor of 2 in the approximation ratio.

A nice feature of this framework is that the algorithms emanating from it, are very simple applications of linear programming and are very simple to analyze. This comes in contrast with the relatively complicated algorithms of [1, 14] whose description/analysis involves a fairly long list of cases. In addition, our framework allows us to tackle more general versions of the problem than those considered in the literature thus far; for example, guards can have weights and we want to minimize the weight of the chosen guards, or where we are not required to cover all the terrain, but only a prescribed fraction of it. It seems that such variants are very difficult to deal with, if one tries to use only geometric techniques as the ones used in [1, 14] for the basic problem. We remark also that, for many geometric set covering problems for which constant factor approximations exist (e.g., covering points in the plane by arbitrary radii disks [3]), it is not clear how to extend these results to the weighted case. So this paper gives one example where such an extension is possible.

It is worth noting that the idea of using the fractional solution to the LP-covering problem to partition the problem into several integral subproblems has been used before [11, 20, 9].

In the next section, we define the basic guarding problem and its variants more formally. In Section 3 we focus on the left guarding problem and show that this is a totally balanced covering problem. Section 4 shows how to get a 2-approximation for one-sided guarding. Finally, in Section 5 we apply these results to obtain constant-factor approximation algorithms for more general variants of the guarding problem.

2. Preliminaries

A *terrain* T is an x-monotone polygonal chain with n vertices, i.e., a piecewise linear curve intersecting any vertical line in at most one point. Denote by V the vertices of T and by $n = |V|$ the complexity of the chain. The terrain polygon P_T determined by T is the closed region in the plane bounded from below by T.

For two points p and q in P_T, we say that p *sees* q and write $p \sim q$, if the line segment connecting p and q is contained in P_T, or equivalently, if it never goes strictly below T. We will also write $p < q$ if p lies to the left of q.

The $1.5D$-terrain guarding problem for T is to place guards on T such that every point $p \in P_T$ is seen by some guard. One can easily see, by the monotonicity of T, that any set of guards that guards T is also enough to guard P_T. Henceforth we restrict our attention to the case when the requirement is to guard all points of T.

The *continuous* $1.5D$-terrain guarding problem is to select a smallest set of guards $A \subseteq T$ that sees every point in T; in other words, for every $p \in T$ there exists $g \in A$ such that $g \sim p$. We also consider the following variants of this basic problem:

(1) In the *discrete* version we are given a set of possible guards $G \subseteq T$ with weights $w : G \to R^+$ and a set of points $N \subseteq T$. The goal is to select a minimum weight set of guards $A \subseteq G$ to guard N.

(2) In the *partial* version we are given a profit function $p : N \to \mathbb{R}^+$ and a budget b. The goal is to find a minimum weight set of guards such that the profit of unguarded points is at most b. In the continuous variant, b is the length of T that can be left unguarded.

(3) In the *one-sided guarding* version the guards can see in only one of two directions: left or right. Specifically, given 3 sets of points N, G_L and G_R, we want to find sets $A_L \subseteq G_L$ and $A_R \subseteq G_R$ of guards such that for all $p \in N$ there is $g \in A_L$ such that $g < p$ and $g \sim p$, or $g \in A_R$ such that $g > p$ and $g \sim p$. The sets G_L and G_R, and hence A_L and A_R need not be disjoint. The overall cost of the solution is $w(A_L) + w(A_R)$.

This includes both the *left-* and *right-guarding* versions where guards in the given set G can see only from the left, respectively, right (setting $G_L = G$ and $G_R = \emptyset$ we get the left-guarding problem, while setting $G_R = G$ and $G_L = \emptyset$ gives the right-guarding problem).

Using a unified framework we get 4-approximations for nearly all[1] of these variants. Our approach is based on linear programming, totally balanced matrices, and the paradigm of rounding to an integral problem [9, 11]. We progressively build our approximations

[1]The only exception is instances of the discrete variant when $G \cap N \neq \emptyset$. Here we get a 5-approximation.

by reducing each variant to a simpler problem. First, we start establishing a connection between the left-guarding problem and totally balanced matrices. Then, we show how to use this to get a 2-approximation for the one-sided guarding. Finally, we show how the latter implies a 4-approximation for other variants.

Throughout the paper we will make frequent use of the following easy-to-prove claim.

Lemma 2.1 ([1])**.** *Let $a < b < c < d$ be four points on T. If $a \sim c$ and $b \sim d$, then $a \sim d$.*

Let $S(p) = \{g \in G \mid g \sim p\}$ be the set of guards that see point $p \in N$. Denote by $S_L(p) = \{g \in G \mid g < p \text{ and } g \sim p\}$ the set of guards that see p strictly from the left, and analogously by $S_R(p)$ the set of guards that see p strictly from the right.

3. Left-guarding and totally balanced matrices

Even though this section deals exclusively with the left-guarding version, it should be noted that everything said applies, by symmetry, to the right-guarding version. Recall in this case that we are given two sets of points N, G, where each point in N has to be guarded using only guards from G that lie strictly to its left.

Consider the following integer linear programming formulation.

$$\text{minimize} \sum_{g \in G} w_g \, x_g \tag{LP1}$$

subject to

$$\sum_{g \in S_L(p)} x_g \geq 1 \qquad \forall p \in N \tag{3.1}$$

$$x_g \in \{0,1\} \qquad \forall g \in G$$

Variable x_g indicates whether g is chosen as a guard. Constraint (3.1) asks that every point is seen by some guard from the left. In the following we will show that the solution of the relaxation of (LP1) will always be integral.

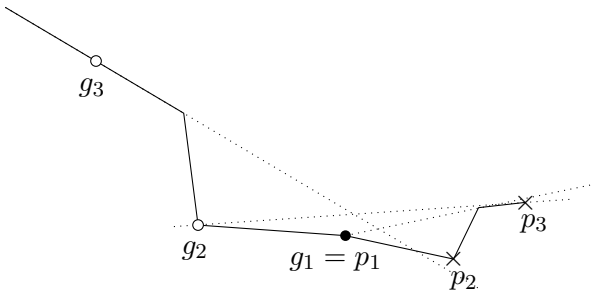
Let $A \in \{0,1\}^{|N| \times |G|}$ be a binary matrix. Call A a *left-visibility* matrix if it corresponds to the guard-point incidence matrix of the coverage problem defined by (LP1) for some instance of the left-guarding problem. Also, A is said to be *totally balanced* [2] if it does not contain a square submatrix with all row and column sums equal to 2 and no identical columns. Finally, A is in standard greedy form if it does not contain as an induced submatrix

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}. \tag{3.2}$$

An equivalent characterization [12] is that A is totally balanced if and only if A can be put into greedy standard form by permuting its rows and columns.

Lemma 3.1. *Any left-visibility matrix is totally balanced.*

Proof. Let A be a left-visibility matrix. We show how to put A into standard greedy form. Permute the rows and columns of A such that the rows from top to bottom correspond to the points ordered from left to right, and the columns from left to right correspond to the guards ordered from right to left. Suppose that there exists an induced 2×2 sub-matrix of the form (3.2), whose rows are indexed by $p_1, p_2 \in N$, and whose columns are indexed by $g_1, g_2 \in G$. Then we have the following order: $g_2 < g_1 < p_1 < p_2$. Now we apply Lemma 2.1 with $a = g_2$, $b = g_1$, $c = p_1$ and $d = p_2$ to arrive at the contradiction $p_2 \sim g_2$. ∎



$$A = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{ccc} g_1 & g_2 & g_3 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{array}$$

Figure 1: Consider the set $G = \{g_1, g_2, g_3\}$ of guards and the set $N = \{p_1, p_2, p_3\}$ of points as above. Visibility matrix A is shown to the right of the example. Note that the guard g_1 guards the point on which it lies, i.e. g_1 guards p_1. Vertices of the polyhedron $\{x \geq 0 \ : \ Ax \geq 1\}$ are $(0, 1, 1)$, $(1, 1, 0)$, $(1, 0, 1)$ and $(1/2, 1/2, 1/2)$.

It is known that for a totally balanced matrix A, the polyhedron $\{x \geq 0 \ : \ Ax \geq 1\}$ is integral. Furthermore, there is an efficient purely combinatorial algorithm for finding an optimal integral solution to (LP1) due to Kolen [16]. Indeed, in the next subsection we show that this algorithm translates into an extremely simple procedure for the uniform weight case, i.e., when $w_g = 1$ for all $g \in G$.

3.1. Uniform left-guarding

For each point $p \in N$ let $L(p)$ denote the left-most guard that sees p. Consider the simple greedy algorithm on the set of points N shown below: points in N are scanned from left to right and when we find an unguarded point p, we select $L(p)$ as a guard.

```
LEFT-GUARDING (T, N, G)
A ← ∅
for p ∈ N processed from left to right
  if p is not yet seen by A then
    A ← A ∪ {L(p)}
return A
```

The algorithm can be implemented in $O(|N| \log |G|)$ time using a procedure similar to Graham's scan [10] for convex-hull computation. To see that it returns an optimal solution, let $X \subseteq N$ be those points that force the algorithm to add a guard. Suppose, for the sake of contradiction, that there exist two points p' and p'' in X that are seen from the left by the same guard $g \in G$, in other words, $g < p' < p''$ and $g \sim p'$ and $g \sim p''$. Let $g' = L(p')$, and note that $g' \leq g$. If $g' = g$ then $g' \sim p''$ and therefore p'' would have not been unguarded when it was processed. Hence $g' < g$, but Lemma 2.1 tells us that $g' \sim p''$ and we get a contradiction. Therefore, each guard in G can see at most one point in X, which means $|X|$ is a lower bound on the optimal solution. Since the cardinality of A equals that of X, it follows that A is optimum, and hence by, Lemma 3.1, it returns an optimal solution of (LP1).

Remark: Note that in our definition of left-guarding or right-guarding the guard does not see the point on which it lies. With the following example (see Figure 1) we demonstrate that without that condition the polyhedron is not necessarily integral any more.

4. A 2-approximation for one-sided guarding

In this section we study discrete weighted one-sided guarding. Recall that in this variant, we are given a set of points N and two sets of guards G_L and G_R, where each guard in G_L (respectively, G_R) can only guard points from N strictly to its right (respectively, strictly to its left). We assume without loss of generality that each point in N can be seen by a guard on its left or by a guard on its right. Otherwise it must be guarded by itself and the system is infeasible, a situation which can be discovered in a preprocessing step.

We state our main result and then describe the algorithm.

Theorem 4.1. *There is a 2-approximation algorithm for discrete one-sided guarding.*

Consider the following LP for finding the optimal set of left and right guards:

$$\text{minimize} \quad \sum_{g\in G_L} w_g\, x_{g,L} + \sum_{g\in G_R} w_g\, x_{g,R} \tag{LP2}$$

$$\text{subject to}$$

$$\sum_{g\in S_L(p)\cap G_L} x_{g,L} \;+\; \sum_{g\in S_R(p)\cap G_R} x_{g,R} \;\geq 1 \qquad \forall p \in N \tag{4.1}$$

$$x_{g,L} \geq 0 \qquad \forall g \in G_L$$

$$x_{g,R} \geq 0 \qquad \forall g \in G_R$$

Variable $x_{g,L}$ indicates whether g is chosen in A_L and $x_{g,R}$ indicates whether g is chosen in A_R. Constraint (4.1) asks that every point is seen by some guard, either from the left or from the right.

The algorithm first finds an optimal fractional solution x^* to (LP2). Guided by x^*, we divide the points into two sets

$$N_L = \left\{ p \in N \mid \textstyle\sum_{g\in S_L(p)\cap G_L} x^*_{g,L} \geq \frac{1}{2} \right\}, \text{ and}$$

$$N_R = \left\{ p \in N \mid \textstyle\sum_{g\in S_R(p)\cap G_R} x^*_{g,R} \geq \frac{1}{2} \right\}.$$

Using the results from Section 3, we solve optimally the left-guarding problem for the pair (N_L, G_L) and the right-guarding problem for the pair (N_R, G_R). This gives us two sets of guards A^*_L and A^*_R. The final solution is a combination of these two.

It is easy to construct examples where solving separately the left-guarding and right-guarding problems and then taking the minimum of these two solutions is arbitrarily far from the optimal value. The intuition behind the algorithm is to use the LP solution to determine which points should be guarded from the left and which should be guarded from the right. The fractional solution also allows us to bound the cost of A^*_L and A^*_R.

Lemma 4.2. *Let A^*_L and A^*_R be optimal solutions for the pairs (N_L, G_L) and (N_R, G_R) respectively. Then $w(A^*_L) \leq 2\sum_{g\in G_L} w_g\, x^*_g$ and $w(A^*_R) \leq 2\sum_{g\in G_R} w_g\, x^*_g$.*

Proof. We only prove the first inequality as the second is symmetrical. Setting $x_{g,L} = 2x^*_g$ we get a fractional solution for (LP1) for guarding N_L. The solution x is feasible, by definition of N_L, and its cost is $2\sum_{g\in G_L} w_g\, x^*_g$. Therefore, the optimal fractional solution can only be smaller than that. Lemma 3.1 tells us that the cost of an optimal fractional solution is the same as the cost of an optimal integral solution, namely, $w(A^*_L)$. ∎

Since $\sum_{g\in G_L} w_g\, x^*_{g,L} + \sum_{g\in G_R} w_g\, x^*_{g,R}$ is a lower bound on the cost of an optimal solution for guarding N, it follows that the cost of (A^*_L, A^*_R) is at most twice the optimum. To see that this is feasible, consider some point $p \in N$. Because of (4.1) and our assumption that each point is seen by some guard on its left or on its right, it must be the case that $p \in N_L$ or $p \in N_R$. Therefore p must be covered, either from the left by A^*_L or from the right by A^*_R.

To compute A^*_L and A^*_R we can take the fractional solution to (LP1) and turn it into a basic, and therefore integral, solution without increasing its cost. Alternatively, we can run Kolen's algorithm [16] for matrices in greedy standard form. This finishes the proof of Theorem 4.1.

4.1. Partial covering

In this section we focus on the partial version of the one-sided guarding problem.

Theorem 4.3. *There is a polynomial $(2+\epsilon)$-approximation and a quasi-polynomial time 2-approximation for partial discrete one-sided guarding.*

Our approach is based on the framework of Mestre [20]. We say A is a *one-sided-visibility* matrix if it is the guard-point incidence matrix of the covering problem defined by (LP2) for some instance of the one-sided guarding problem. Also, A is said to be *2-separable* if there exist binary matrices A_1 and A_2 such that $A = A_1 + A_2$ and every matrix B formed by taking rows from A_1 or A_2 is totally balanced (the ith row of B is the ith row of A_1 or the ith row of A_2, for all i).

Proposition 4.4 ([20])**.** *Let A be a 2-separable matrix. Then there is a $(2+\epsilon)$-approximation and a quasi-polynomial time 2-approximation for the partial problem defined by A.*

Therefore, all we need to do to prove Theorem 4.3 is to argue that every one-sided visibility matrix is 2-separable.

Lemma 4.5. *Any one-sided visibility matrix is 2-separable.*

Proof. Let A be a one-sided visibility matrix and assume, without loss of generality, that A has the form $[C_1\ C_2]$ where the columns of C_1 correspond to left guards G_L and the columns of C_2 correspond to the right guards G_R.

Our decomposition of A uses $A_1 = [C_1\ 0]$ and $A_2 = [0\ C_2]$. Suppose that a matrix B is formed by taking rows from A_1 and A_2. Let N_L be the set of rows originating from A_1 and N_R the set of rows originating from A_2 (note that N_L and N_R constitute a partition of N). Permute the rows of B so that rows in N_L appear before rows in N_R. This gives rise to the following block matrix

$$B' = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$$

where the rows of D_1 correspond to points in N_L and its columns to left guards, and the rows of D_2 correspond to points in N_R and its columns to right guards. By Lemma 3.1

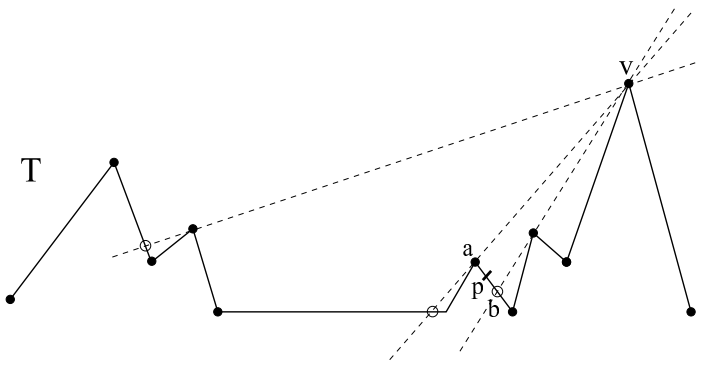


Figure 2: Example of additional set of points/guards for a vertex v of T. Point p is the point selected from the essential segment $\overline{ab}$.

both D_1 and D_2 are totally balanced. Therefore we can permute the rows and columns of B' to get a new matrix

$$B'' = \begin{bmatrix} D_1' & 0 \\ 0 & D_2 \end{bmatrix}$$

where D_1' and D_2' are in standard greedy form, which in turn implies that B'' is also in standard greedy form. It follows that B'', B', and B are totally balanced. ∎

This finishes the proof of Theorem 4.3.

5. Applications

In this section we show how to use the 2-approximations for one-sided guarding to design good approximation algorithms for more general variants.

5.1. The continuous case

We assume that the weights are uniform[2]. Recall that in this variant guards can be placed anywhere on the terrain and we are to guard all the points. We will reduce the problem to to the discrete case, where $G \cap N = \emptyset$. Our reduction follows the approach of Ben-Moshe et al. [1].

Theorem 5.1. *There is a 4-approximation algorithm for the continuous case and and a $(4+\epsilon)$-approximation for its partial version.*

Let A^* be an optimum set of guards for a given instance T of the continuous problem. Consider a guard g in A^*. If g is not a vertex of T then it must lie on a segment $\overline{pq}$ of T. Suppose without loss of generality that $p < q$, then a left guard at p and a right guard at q can see at least as much as g does. If g is a vertex of T then a left guard and a right guard at g together can see the same as g does minus g itself. Therefore there exists a solution A' that uses only left and right guards on the vertices of T that covers $T \setminus V$ such that $|A'| = 2|A^*|$.

[2]This assumption can be removed using standard discretization techniques at the expense of a small increase in the approximation factor

To deal with the fact that every point must be guarded, consider the line through each pair of vertices $v_1, v_2 \in V$ such that v_1 *see also* v_2 and introduce at most two new points that see v_1 and v_2 at the place where the line intersects the terrain. These points partition T into $O(n^2)$ *essential segments*. In the strict interior of each segment introduce an additional point p that is *responsible* for the segment. Let M be the set of all such points. (See Figure 2 for an illustration.) The key realization is that for every guard $g \in V$ and essential segment $\overline{ab}$, either g can see the whole segment or nothing of it.

Hence, a feasible solution to the one-sided discrete version with $G_L = G_R = V$ and $N = M$ also constitutes a feasible solution to the continuous case. Let A'' be an optimal solution for this discrete problem, and A''' be the solution returned by Theorem 4.1. Since A' is feasible for the discrete instance, we get $|A'''| \leq 2|A''| \leq 2|A'| = 4|A^*|$ and we get an overall approximation factor of 4.

For the partial version where we want at most a fraction of the length to be left unguarded we give to each point in $p \in M$ a profit equal to the length of the essential segment it is responsible for.

5.2. The discrete case

We consider the discrete version where we are given a set of guards G and set of points N to guard. In this case, guards can see in both directions.

Theorem 5.2. *There is a* 4*-approximation for the weighted discrete case and* $(4+\epsilon)$*-approximation for its partial version when* $G \cap N = \emptyset$*. Otherwise, we get* 5 *and* $(5+\epsilon)$*-approximations respectively.*

The case where $G \cap N = \emptyset$ is easily handled by replacing a guard that can see in both directions with a left guard and a right guard. Thus we pay a factor 2 to reduce the general problem to one-sided guarding. This also holds for the partial version.

Notice that if $G \cap N \neq \emptyset$ then the reduction above must pay a factor of 3 since a point guarding itself must be guarded by some other point strictly from the left or the right, and thus it only leads to a 6-approximation. To get the ratio of 5 we need to use yet another linear program.

$$\begin{aligned} & \text{minimize} && \sum_{g \in G} w_g \, x_g && && \text{(LP3)} \\ \text{subject to} & && && \\ & \sum_{g \in S(p)} x_g \ \geq 1 && && \forall p \in N \\ & x_g \ \geq 0 && && \forall g \in G \end{aligned}$$

Let x^* be an optimal fractional solution to (LP3). As in the one-sided case we will let the solution x dictate which points should be self-guarded and which should be guarded by others. Define

$$A_0 = \left\{ g \in N \cap G \mid x^*_g \geq \tfrac{1}{5} \right\}.$$

We place guards at A_0 at a cost of most

$$w(A_0) \leq 5 \sum_{g \in A_0} w_g x^*_g. \tag{5.1}$$

Let N' be the set of points in N not seen by A_0 and let $G' = G \setminus A_0$. We will construct a fractional solution for the one-sided guarding problem for N' and $G_L = G_R = G'$. For each $g \in G'$ let $x_{g,L} = x_{g,R} = \frac{5}{4}x^*_g$. The fractional solution x is feasible for (LP2) since for all $p \in N'$

$$\sum_{g \in S_L(p) \cap G_L} x_{g,L} \; + \sum_{g \in S_R(p) \cap G_R} x_{g,R} = \frac{5}{4} \sum_{g \in S(p) \setminus \{p\}} x^*_g \geq \frac{5}{4}\left(1 - \frac{1}{5}\right) = 1,$$

Let (A^*_L, A^*_R) be the solution found for the one-sided problem. The cost of these sets of guards is guaranteed to be at most twice that of x, which in turn is $\frac{5}{2}\sum_{g \in G \setminus A_0} w_g x^*_g$. Thus the overall cost is

$$w(A^*_L) + w(A^*_R) \leq 5 \sum_{g \in G \setminus A_0} w_g x^*_g. \tag{5.2}$$

Hence, the second part of Theorem 5.2 follows from (5.1) and (5.2). Finally, for the partial version we note the proof of Proposition 4.4 uses as a lower bound the cost of the optimal fractional solution, so the cost of the solution returned can still be related to the cost of x, which is necessary to get the stated approximation guarantee.

6. Concluding remarks

We gave a 4-approximation for the continuous 1.5D terrain guarding problem as well as several variations of the basic problem. Our results rely, either explicitly or implicitly, on the LP formulation (LP3) for the discrete case. For the unweighted version of the problem, there is a very simple $O(|N| \log |G|)$-algorithm for solving the left-guarding LP (LP1). Furthermore, at the loss of factor $(1 + \epsilon)$ in the approximation ratios, one can use fast techniques for covering LPs (see e.g. [8, 21]) to solve (LP2). In particular, using the recent results of [17, 18], a $(1 + \epsilon)$-approximation for (LP2) can be found in time $O(|N||G| \log(|N| + |G|)/\epsilon^2)$, which is only a polylogarithmic factor slower than the purely combinatorial algorithms for the uniform weight case (see e.g. [14]).

Very recently, King [15] showed that the VC dimension of the discrete case is exactly 4. More precisely, he showed a terrain with 4 guards and 16 points (these sets are disjoint) such that each point is seen by a different subset of the guards. If we have to cover the points that are seen by pairs of guards, we get precisely a vertex cover problem on the complete graph with 4 vertices. An integral solution must pick 3 vertices, while a fractional solution can pick a half of all vertices. It follows that the integrality gap of (LP3) is at least 3/2, even when $G \cap N = \emptyset$. On the other hand, our analysis shows that the gap is at most 4. We leave as an open problem to determine the exact integrality gap of (LP3).

Acknowledgements

We would like to thank Jamie King, Kasturi Varadarajan, and the anonymous referees for their valuable comments. The forth author was supported by an Alexander von Humboldt Fellowship.

References

[1] B. Ben-Moshe, M. J. Katz, and J. S. B. Mitchell. A constant-factor approximation algorithm for optimal 1.5D terrain guarding. *SIAM Journal on Computing*, 36(6):1631–1647, 2007.

[2] C. Berge. Balanced matrices. *Mathematical Programming*, 2:19–31, 1972.

[3] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.

[4] D. Z. Chen, V. Estivill-Castro, and J. Urrutia. Optimal guarding of polygons and monotone chains. In *Proceedings of the 7th Canadian Conference on Computational Geometry*, pages 133–138, 1995.

[5] K. L. Clarkson and K. R. Varadarajan. Improved approximation algorithms for geometric set cover. In *Proceedings of the 20th Symposium on Computational Geometry*, pages 135–141, 2005.

[6] E. D. Demaine and J. O'Rourke. Open problems: Open problems from CCCG 2005. In *Proceedings of the 18th Canadian Conference on Computational Geometry*, pages 75–80, 2006.

[7] K. Elbassioni, D. Matijević, J. Mestre, and D. Ševerdija. Improved approximations for guarding 1.5-dimensional terrains. *CoRR*, abs/0809.0159v1, 2008.

[8] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 300–309, 1998.

[9] 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.

[10] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[11] R. Hassin and D. Segev. Rounding to an integral program. *Operations Research Letters*, 36(3):321–326, 2008.

[12] A. J. Hoffman, A. Kolen, and M. Sakarovitch. Totally-balanced and greedy matrices. *SIAM Journal on Algebraic and Discrete Methods*, 6:721–730, 1985.

[13] J. King. Errata on "A 4-Approximation Algorithm for Guarding 1.5-Dimensional Terrains". `http://www.cs.mcgill.ca/~jking/papers/4apx_latin.pdf`.

[14] J. King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In *Proceedings of the 13th Latin American Symposium on Theoretical Informatics*, pages 629–640, 2006.

[15] J. King. VC-dimension of visibility on terrains. In *Proceedings of the 20th Canadian Conference on Computational Geometry*, pages 27–30, 2008.

[16] A. Kolen. *Location problems on trees and in the rectilinear plane*. PhD thesis, Matematisch Centrum, Amsterdam, 1982.

[17] C. Koufogiannakis and N. E. Young. Beating simplex for fractional packing and covering linear programs. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 494–504, 2007.

[18] C. Koufogiannakis and N. E. Young. Beating simplex for fractional packing and covering linear programs. *CoRR*, abs/0801.1987, 2008.

[19] E. Krohn. Survey of terrain guarding and art gallery problems. Unpublished manuscript. November 2007.

[20] J. Mestre. Lagrangian relaxation and partial cover (extended abstract). In *Proceedings of the 25th Annual Symposium on Theoretical Aspects of Computer Science*, pages 539–550, 2008.

[21] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 495–504, 1991.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 373–384
www.stacs-conf.org

COVER TIME AND BROADCAST TIME

ROBERT ELSÄSSER [1] AND THOMAS SAUERWALD [2]

[1] Institute for Computer Science, University of Paderborn, 33102 Paderborn, Germany
E-mail address: elsa@upb.de

[2] International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704, U.S.
E-mail address: sauerwal@icsi.berkeley.edu

Abstract. We introduce a new technique for bounding the cover time of random walks by relating it to the runtime of randomized broadcast. In particular, we strongly confirm for dense graphs the intuition of Chandra et al. [8] that "the cover time of the graph is an appropriate metric for the performance of certain kinds of randomized broadcast algorithms". In more detail, our results are as follows:

- For any graph $G = (V, E)$ of size n and minimum degree δ, we have $\mathcal{R}(G) = \mathcal{O}(\frac{|E|}{\delta} \cdot \log n)$, where $\mathcal{R}(G)$ denotes the quotient of the cover time and broadcast time. This bound is tight for binary trees and tight up to logarithmic factors for many graphs including hypercubes, expanders and lollipop graphs.
- For any δ-regular (or almost δ-regular) graph G it holds that $\mathcal{R}(G) = \Omega(\frac{\delta^2}{n} \cdot \frac{1}{\log n})$. Together with our upper bound on $\mathcal{R}(G)$, this lower bound strongly confirms the intuition of Chandra et al. for graphs with minimum degree $\Theta(n)$, since then the cover time equals the broadcast time multiplied by n (neglecting logarithmic factors).
- Conversely, for any δ we construct almost δ-regular graphs that satisfy $\mathcal{R}(G) = \mathcal{O}(\max\{\sqrt{n}, \delta\} \cdot \log^2 n)$. Since any regular expander satisfies $\mathcal{R}(G) = \Theta(n)$, the strong relationship given above does not hold if δ is polynomially smaller than n.

Our bounds also demonstrate that the relationship between cover time and broadcast time is much stronger than the known relationships between any of them and the mixing time (or the closely related spectral gap).

1. Introduction

Motivation. A *random walk* on a graph is the following process. Starting from a specified vertex, the walk proceeds at each step from its current position to an adjacent vertex chosen uniformly at random. The study of random walks has numerous applications in the design and analysis of algorithms (cf. [24] for a survey). Two of the most important

1998 ACM Subject Classification: G.3 Probability and Statistics [Probabilistic Algorithms, Stochastic Processes].

Key words and phrases: Random walk, randomized algorithms, parallel and distributed algorithms.

This work has been partially supported by the IST Program of the European Union under contract number 15964 (AEOLUS), by the German Science Foundation (DFG) Research Training Group GK-693 of the Paderborn Institute for Scientific Computation (PaSCo) and by the German Academic Exchange Service (DAAD).

parameters of random walks are its *mixing time* which is the time until the walk becomes close to the stationary distribution, and its *cover time* which is the expected time required for the random walk to visit all vertices.

Famous combinatorial problems solved by rapidly mixing random walks are, e.g., approximating the permanent and approximating the volume of convex bodies (cf. [24] for more details). The cover time comes naturally into play when the task is to explore a network, or to estimate the stationary distribution of a graph [31]. Moreover, the cover time is intimately related to combinatorial and algebraic properties such as the conductance and the spectral gap of the underlying graph [6] and thus, bounding the cover time may also lead to interesting combinatorial results.

In this paper, we are particularly interested in the relationship between the cover time of random walks and the runtime of *randomized broadcast* [16]. Broadcasting in large networks has various fields of application in distributed computing such as the maintenance of replicated databases or the spreading of information in networks [16, 21]. Furthermore it is closely related to certain mathematical models of epidemic diseases where infections are spread to some neighbours chosen uniformly at random with some probability. However, in most papers, spreaders are only active in a given time frame, and the question of interest is, whether on certain networks an epidemic outbreak occurs [22, 27]. Several threshold theorems involving the basic reproduction number, contact number, and the replacement number have been stated (see [19] for a collection of results).

Here, we consider the so-called *randomized broadcast algorithm* [16] (also known as *push algorithm*): at the beginning, a vertex s in a graph G knows of some rumor which has to be disseminated to all other vertices. Then, at each time-step every vertex that knows of the rumor chooses one of its neighbors uniformly at random and informs it of the rumor. The advantage of *randomized* broadcast is in its inherent robustness against several kinds of failures (e.g., [16]) and dynamical changes compared to deterministic schemes that either need substantially more time or can tolerate only a relatively small number of faults [21].

Related Work. There is a vast body of literature devoted to the cover time of random walks and we can only point to some results directly related to this paper. Aleliunas et al. [3] initiated the study of the cover time. Amongst other results, they proved that the cover time of any graph $G = (V, E)$ with n vertices is at most $\mathcal{O}(n \cdot |E|)$. To obtain this result they proved that the cover time is bounded by the weight of a spanning tree whose edges are weighted according to the commute times between the corresponding vertices. This approach was later refined by Feige [15] to obtain an upper bound of less than $2n^2$ for regular graphs. While the spanning tree technique is particularly useful for graphs that have a high cover time [15], it vastly overestimates the cover time of e.g., complete graphs.

The seminal work of Chandra et al. [8] established a close connection between the *electrical resistance* of a graph and its cover time. This correspondence allows the application of elegant methods from electrical network theory, e.g., the use of short-cut-principles or certain flow-based arguments. Nevertheless, for the computation of the resistance of a given graph other graph-theoretical parameters are often required, e.g., vertex-expansion, number of vertex-disjoint paths or the number of vertices within a certain distance [8].

A wide range of techniques to upper bound the cover time is based on the mixing time of a random walk or the closely related spectral gap. The technique of reducing the cover time to the coupon collector's problem on graphs with low mixing time traces back to Aldous [1] who derived tight bounds on the cover time of certain Cayley graphs. Later, Cooper and Frieze extended this technique to bound the cover time of several classes of random graphs,

e.g., [10]. The basic idea of this method is that after each mixing time steps, the random walk visits an (almost) randomly chosen vertex. The crux is to deal with the dependencies among the intermediate vertices. Hence, in addition to an upper bound on the mixing time of logarithmic [10] or at least sub-polynomial order [1], one has to bound the number of returns to the starting vertex within mixing time steps.

A related result was derived by Broder and Karlin [6] who bounded the cover time in terms of the *spectral gap* $1-\lambda_2$, where λ_2 is the second largest eigenvalue of the transition matrix of the random walk.

Winkler and Zuckerman [31] introduced an interesting parameter called *blanket time* which is closely related to the cover time. Here, one asks for the first time-step at which the observed distribution of the visited vertices approximates the stationary distribution up to a constant factor. Winkler and Zuckerman conjectured that the blanket time is asymptotically the same as the cover time. In [20] Kahn et al. showed that the blanket time is upper bounded by the cover time multiplied by $\mathcal{O}((\ln\ln n)^2)$ for any graph.

Most papers dealing with randomized broadcast analyze the runtime on different graph classes. Pittel [28] proved that the runtime on complete graphs is $\log_2 n + \ln n \pm \mathcal{O}(1)$. Feige et al. [16] derived several upper bounds, in particular a bound of $\mathcal{O}(\log n)$ for hypercubes and random graphs. We extended the bound of $\mathcal{O}(\log n)$ to a certain class of Cayley graphs in [12]. Additionally, we proved that the broadcast time is upper bounded by the sum of the mixing time and an additional logarithmic factor [29] (a similar result for a related broadcast algorithm was derived by Boyd et al. [5]). However, the mixing time cannot be used for an appropriate lower bound on the broadcast time, as it may overestimate the broadcast time up to a factor of n on certain graphs (cf. Section 3.2).

Our Results. We present the first formal results relating the cover time to the broadcast time. In most of them, we will assume that the broadcast and the random walk both start from its respective worst-case initial vertex. Note that at a first look these processes seem not to be too closely related, since randomized broadcast is a *parallel* process where propagation occurs at *every* informed vertex simultaneously, while a random walk moves "only" from *one* vertex to another [16]. Nevertheless, Chandra et al. [8] mentioned that "The cover time of the graph is an appropriate metric for the performance of certain kinds of randomized broadcast algorithms". As a consequence of our main results, we obtain a fairly tight characterization of graph classes for which the cover time and broadcast time capture each other. On the positive side, for every graph with minimum degree $\Theta(n)$, the cover time equals the broadcast time multiplied by n, up to logarithmic factors (this kind of tightness (up to logarithmic factors) has been frequently considered in the study of random walks, e.g., when studying rapidly mixing Markov chains [30], or when bounding the cover time [8],[24, Theorem 2.7].). On the negative

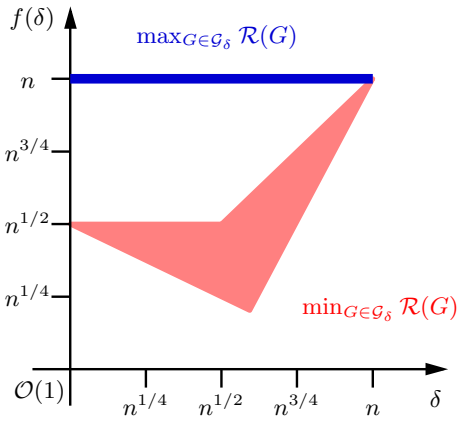


Figure 1: All bounds on $\mathcal{R}(G)$ at a glance. $\mathcal{G}_\delta$ denotes the class of graphs with $\Delta = \mathcal{O}(\delta)$. The blue and red polygons indicate the gap between our lower and upper bounds on $\max_{G\in\mathcal{G}_\delta}\mathcal{R}(G)$ and $\min_{G\in\mathcal{G}_\delta}\mathcal{R}(G)$, resp.

side, this strong correspondence does not hold on almost regular graphs, when the degree is substantially smaller than n.

In more detail, our results illustrated in Figure 1 are as follows. First, we prove that the cover time of any graph with minimum degree δ is at most $\mathcal{O}(\frac{|E|}{\delta}\log n)$ multiplied by the (expected) broadcast time, that is, the quotient $\mathcal{R}(G)$ of the cover time and broadcast time is $\mathcal{O}(\frac{|E|}{\delta}\log n)$. This bound is tight up to a constant factor for binary trees and tight up to a logarithmic factor for various graphs including, e.g., expanders, hypercubes and lollipop-graphs. As an application, we use this result to upper bound the cover time of generalized random graphs that are used as a model for real world networks [9].

Conversely, we consider the question of lower bounding $\mathcal{R}(G)$. By showing that the commute time between two vertices u, v is at least $2 \cdot \mathrm{dist}(u,v)^2$, we obtain that $\mathcal{R}(G) = \Omega(\frac{\sqrt{n}\log n}{\Delta})$ for any graph with maximum degree Δ. For constant Δ, this bound is tight for the two-dimensional $\sqrt{n} \times \sqrt{n}$-torus up to logarithmic factors. We move on to improve this bound for denser graphs with $\Delta = \mathcal{O}(\delta)$ to $\mathcal{R}(G) = \Omega(\frac{\sqrt{n}}{\sqrt{\delta}\log n})$. More importantly, for any graph with $\Delta = \mathcal{O}(\delta)$ we establish that $\mathcal{R}(G) = \Omega(\frac{\delta^2}{n} \cdot \frac{1}{\log n})$. Together with our upper bound on $\mathcal{R}(G)$, this implies that on any graph with $\delta = \Theta(n)$, cover time and broadcast time (multiplied by n) capture each other up to logarithmic factors.

We complement these positive results by the construction of (almost) d-regular graphs for which $\mathcal{R}(G) = \mathcal{O}(\max\{\sqrt{n}, d\} \cdot \log n)$. Since for any d-regular expander (graphs for which the spectral gap satisfies $(1-\lambda_2)^{-1} = \mathcal{O}(1)$), $\mathcal{R}(G) = \Theta(n)$, the cover time does *not* always capture the performance of randomized broadcast for the class of almost d-regular graphs when d is polynomially smaller than n.

All of our lower and upper bounds reveal a surprisingly close relationship between the cover time and broadcast time. In particular, upper bounding the cover time in terms of the broadcast time turns out to be as good as (and in some cases much better than) bounding it in terms of the spectral-gap (cf. Section 3.2). From another perspective, we derive a lower bound on the broadcast time in terms of the cover time that nicely complements the existing upper bounds on the broadcast time based on the mixing time [12, 29]. A further novel feature of this work is the use of techniques from electrical network theory to bound the broadcast time. We should note that certain difficulties in applying such methods for the study of randomized broadcast have been mentioned by Feige et al. [16].

2. Notations, Definitions and Preliminaries

Throughout this paper, let $G = (V, E)$ be an undirected, simple and connected graph of size $n = |V|$. By δ and Δ we denote the minimum and maximum degree of G, respectively. For some set $X \subseteq V$, $N(X)$ denotes the set of all neighbors of $x \in X$, and $\deg_X(u)$ is the number of edges between u and the vertices of X.

Random Walk. A *random walk* [24] on a graph G starts at a specified vertex $s \in V$ and moves in each step to a neighboring vertex chosen uniformly at random. This can be described by a *transition matrix* $\mathbf{P}$, where $p_{ij} = 1/\deg(i)$ if $\{i,j\} \in E(G)$, and $p_{ij} = 0$ otherwise. Then, the random walk is an infinite sequence of vertices $X_0, X_1, \ldots$, where $X_0 := s$ is the starting point of this random walk, and X_t denotes the vertex visited by the random walk at step t. Note that X_t is a random variable with a distribution $\mathbf{p}_s(t)$ on $V(G)$. Denoting by $\mathbf{p}_s(0)$ the unit-vector (regarded as column vector) with 1 at the component corresponding to s and 0 otherwise, we obtain the iteration $\mathbf{p}_s(t+1) = \mathbf{p}_s(t) \cdot \mathbf{P}$

for every step $t \in \mathbb{N}$. It is well-known that on non-bipartite graphs, $\mathbf{p}_s(t)$ converges for $t \to \infty$ towards the *stationary distribution* vector π given by $\pi(v) = \deg(v)/(2|E|)$. For simplicity, we confine ourselves to non-bipartite graphs in the following. This causes no loss of generality as for general graphs (including bipartite ones) convergence can be ensured easily by using the transition matrix $\frac{1}{2}\mathbf{I} + \frac{1}{2}\mathbf{P}$ (with $\mathbf{I}$ being the identity matrix) instead of $\mathbf{P}$. This change of the transition matrix slows down the mixing time (and the cover time) only by some constant factor [24, 30].

Mixing Time and Spectral Gap. The *mixing time* of a random walk on G is $\mathsf{MIX}_\varepsilon(G) := \max_{s\in V} \min\{t \in \mathbb{N} : \ \|\mathbf{p}_s(t) - \pi\|_1 \leqslant \varepsilon, X_0 = s\}$. Since G is connected and non-bipartite, the eigenvalues of $\mathbf{P}$ satisfy $\lambda_1 = 1 > \lambda_2 \geqslant \cdots \geqslant \lambda_n > -1$. The following result by Sinclair shows that the spectral gap $1 - \lambda_2$ captures the mixing time up to logarithmic factors.

Theorem 2.1 ([30])**.** *For any graph $G = (V, E)$ and $\varepsilon > 0$,*

$$\Omega\left(\frac{\lambda_2}{1-\lambda_2} \cdot \log(\frac{1}{\varepsilon})\right) = \mathsf{MIX}_\varepsilon(G) = \mathcal{O}\left(\frac{1}{1-\lambda_2} \cdot \left(\log n + \log(\frac{1}{\varepsilon})\right)\right).$$

Commute Time, Resistance and Cover Time. For two vertices $u, v \in V(G)$, the *hitting time* from u to v is defined as $\mathsf{H}(u, v) := \mathbf{E}\,[\,\min\{t \in \mathbb{N}\backslash\{0\} : X_t = v, X_0 = u\}\,]$, i. e., the expected number of steps to reach v from u. The *commute time* $\mathsf{C}(u, v)$ is defined as the expected number of steps to reach v when starting from u and then returning back to u, so, $\mathsf{C}(u, v) := \mathsf{H}(u, v) + \mathsf{H}(v, u)$. Consider now the graph G as an electrical network where each edge represents a unit resistance. Let u and v be two vertices. Assume that one ampere were injected into vertex u and removed from vertex v. Then $\mathsf{R}(u, v)$ is the voltage difference between u and v (for more details on electrical networks we refer the reader to [8, 24]), and is related to $\mathsf{C}(u, v)$ as follows.

Theorem 2.2 ([8])**.** *For any pair of vertices $u, v \in V$, $\mathsf{C}(u, v) = 2|E| \cdot \mathsf{R}(u, v)$.*

We will mainly be concerned with the *cover time*, which is the expected number of steps a random walk takes to visit all vertices of G. Denote by $\mathsf{COV}(s)$ this time for a random walk which starts from s, and let $\mathsf{COV}(G) := \max_{s\in V} \mathsf{COV}(s)$. The cover time is related to the maximum commute time by means of $\frac{1}{2} \cdot \max_{u,v\in V} \mathsf{C}(u, v) \leqslant \mathsf{COV}(G) \leqslant e^3 \cdot \max_{u,v\in V} \mathsf{C}(u, v) \ln n + n$ [8]. We restate the following bounds by Feige.

Theorem 2.3 ([13, 14])**.** *For any graph, $(1 - o(1)) \cdot n \ln n \leqslant \mathsf{COV}(G) \leqslant (\frac{4}{27} + o(1)) \cdot n^3$.*

A corresponding result to Theorem 2.1 for $\mathsf{COV}(G)$ was given by Broder and Karlin.

Theorem 2.4 ([6])**.** *For any regular graph $G = (V, E)$, $\mathsf{COV}(G) = \mathcal{O}(\frac{1}{1-\lambda_2} \cdot n \log n)$.*

Randomized Broadcast. We will consider the relationship between the cover time of random walks and the following *randomized broadcast algorithm* RBA (also known as push algorithm). Assume that at time $t = 0$ a vertex s knows of a rumor which has to be spread to all other vertices. Then, at each time-step $t = 1, 2, \ldots$ every vertex that knows of the rumor chooses a neighbor uniformly at random and informs it of the rumor. Let I_t be the set of informed vertices at time t, so $I_0 = \{s\}$. The runtime of RBA is denoted by $\mathsf{RBA}_p(G) := \max_{s\in V} \min\{t \in \mathbb{N} : \mathbf{Pr}\,[\,I_t = V \mid I_0 = \{s\}\,] \geqslant 1 - p\}$ for some given $0 < p < 1$. The expected runtime is $\mathbf{E}\,[\,\mathsf{RBA}(G)\,] := \max_{s\in V}\{\mathbf{E}\,[\,\min\{t \in \mathbb{N} : I_t = V,\, I_0 = \{s\}\}\,]\}$. By standard arguments, we have $\mathbf{E}\,[\,\mathsf{RBA}(G)\,] = \mathcal{O}(\mathsf{RBA}_{n^{-1}}(G)) = \mathcal{O}(\mathbf{E}\,[\,\mathsf{RBA}(G)\,] \cdot \log n)$. We remark that $\mathsf{RBA}(G)$ is at least $\max\{\log_2 n, \mathrm{diam}(G)\}$ on any graph G, and $\mathsf{RBA}_{n^{-1}}(G)$ may range from $\Theta(\log n)$ (which is the case for many "nice" graphs) to $\Theta(n \log n)$ (which is the

case for the star) [16]. Sometimes we also use $\mathsf{RBA}(s,v) := \min\{t \in \mathbb{N} : v \in I(t) \mid I(0) = \{s\}\}$ and $\mathsf{RBA}_p(s,v) := \min\{t \in \mathbb{N} : \mathbf{Pr}[\, v \in I(t) \mid I_0 = \{s\}\,] \geqslant 1-p\}$ for some specified $0 < p < 1$. We will frequently make use of following upper bound of Feige et al. [16].

Theorem 2.5 ([16])**.** *For any graph $G = (V,E)$, $\mathsf{RBA}_{n^{-1}}(G) = \mathcal{O}(\Delta \cdot (\log n + \mathrm{diam}(G))$.*

To compare the cover time with the broadcast time, we define $\mathcal{R}(G) := \frac{\mathsf{COV}(G)}{\mathbf{E}[\,\mathsf{RBA}(G)\,]}$.

3. Upper Bound on $\mathcal{R}(G)$ and Applications

3.1. Upper Bound on $\mathcal{R}(G)$

To prove an upper bound on $\mathcal{R}(G)$, we first prove a general inequality between *first-passage-percolation* times and broadcast times and apply then a result of Lyons et al. [25] relating first-passage-percolation to the cover time.

Definition 3.1 ([17, 25])**.** The *undirected first-passage-percolation* UFPP is defined as follows. All (undirected) edges $e \in E(G)$ are assigned weights $w(e)$ that are independent exponential random variable with parameter 1. Specify a vertex s. Then the *first-passage-percolation time* from s to v is defined by $\mathsf{UFPP}(s,v) := \inf_{\mathcal{P}=(s,\ldots,v)} \sum_{e\in\mathcal{P}} w(e)$, where the inf is over all possible paths from s to v in G. Note that $\mathsf{UFPP}(s,s) = 0$.

Theorem 3.2. *For any graph $G = (V,E)$ and $s, v \in V$, $\mathbf{E}[\,\mathsf{UFPP}(s,v)\,] \leqslant \frac{2}{\delta} \cdot \mathbf{E}[\,\mathsf{RBA}(s,v)\,]$.*

Proof. In the proof we derive several (in-)equalities between different percolation and broadcast models. First we introduce a directed version of UFPP, denoted by DFPP. In this model each undirected edge $\{u,u'\} \in E(G)$ is replaced by two directed edges (u,u') and (u',u), and all directed edges e are assigned weights $w(e)$ that are independent exponential random variable with parameter 1. Denote by $\mathsf{DFPP}(s,v)$ the corresponding first-passage-percolation time of this directed version.

Lemma 3.3. *For any graph $G = (V,E)$ and $s, v \in V$, $\mathbf{E}[\,\mathsf{UFPP}(s,v)\,] \leqslant 2 \cdot \mathbf{E}[\,\mathsf{DFPP}(s,v)\,]$.*

Next consider another broadcast model denoted by $\overline{\mathsf{SEQ}}$. At the beginning, a vertex s knows of a rumor which has to be spread to all other vertices. Once a vertex u receives the rumor at time $t \in \mathbb{R}$, it sends the rumor at each time $t + X_{1,u}$, $t + X_{1,u} + X_{2,u}, \ldots$ to a randomly chosen neighbor, where the $X_{i,u}$ with $i \in \mathbb{N}$ are independent exponential variables with parameter $\deg(u)$. Let $\overline{\mathsf{SEQ}}(s,u)$ be the first time when u is informed.

Lemma 3.4. *For any $s, v \in V$, $\overline{\mathsf{SEQ}}(s,v)$ and $\mathsf{DFPP}(s,v)$ have the same distribution.*

Finally, our aim is to relate $\overline{\mathsf{SEQ}}$ and RBA.

Observation 3.5. In any execution of RBA, there is for each $v \in V$ at least one minimal path $\mathcal{P}_{\min}(s,v) = (s = v_0 \overset{D_1}{\to} v_1 \overset{D_2}{\to} \ldots \overset{D_{l-1}}{\to} v_l = v)$, such that for each i, v_i sends the rumor v_{i+1} at time $\mathsf{RBA}(s,v_i) + D_{i+1}$, and at this time v_{i+1} becomes informed for the first time.

Using this observation and a coupling argument, we can prove the following lemma.

Lemma 3.6. *For any pair of vertices $s, v \in V$ we have $\mathbf{E}\left[\,\overline{\mathsf{SEQ}}(s,v)\,\right] \leqslant \frac{\mathbf{E}[\,\mathsf{RBA}(s,v)\,]}{\delta}$.*

We are now ready to finish the proof of Theorem 3.2. For every pair of vertices $s, v \in V$,

$$\mathbf{E}[\,\mathsf{UFPP}(s,v)\,] \leqslant 2 \cdot \mathbf{E}[\,\mathsf{DFPP}(s,v)\,] = 2 \cdot \mathbf{E}\left[\,\overline{\mathsf{SEQ}}(s,v)\,\right] \leqslant \frac{2}{\delta} \cdot \mathbf{E}[\,\mathsf{RBA}(s,v)\,].$$

■

Theorem 3.7 ([25]). *Let $s, v \in V(G)$ with $s \neq v$. Then,* $\mathsf{R}(s,v) \leqslant \mathbf{E}[\,\mathsf{UFPP}(s,v)\,]$.

Combining the two theorems above we arrive at the main result of this section.

Theorem 3.8. *For any graph $G = (V, E)$ we have for every pair of vertices $s \neq v$,*

$$\mathsf{C}(s,v) \leqslant 4 \cdot \frac{|E|}{\delta} \cdot \mathbf{E}[\,\mathsf{RBA}(s,v)\,],$$

and hence $\mathsf{COV}(G) = \mathcal{O}\left(\frac{|E|}{\delta} \cdot \log n \cdot \mathbf{E}[\,\mathsf{RBA}(G)\,]\right)$ *or equivalently,* $\mathcal{R}(G) = \mathcal{O}\left(\frac{|E|}{\delta} \cdot \log n\right)$.

3.2. Applications

We start by giving examples for which the first inequality of Theorem 3.8 is asymptotically tight. For paths and cycles with n vertices, it is well-known that $\max_{s,v} \mathsf{C}(s,v) = \Theta(n^2)$ (e.g., [24]) and Theorem 2.5 gives $\max_{s,v} \mathbf{E}[\,\mathsf{RBA}(s,v)\,] \leqslant \mathbf{E}[\,\mathsf{RBA}(G)\,] = \mathcal{O}(n)$. Similarly, for lollipop graphs (a complete graph with $2n/3$ vertices attached by a path of length $n/3$), $\max_{s,v} \mathsf{C}(s,v) = \Theta(n^2)$ (e.g., [24]) and $\mathbf{E}[\,\mathsf{RBA}(G)\,] = \mathcal{O}(n)$, and therefore the first inequality of Theorem 3.8 is also asymptotically tight for this highly non-regular graph.

The following overview in Figure 3.2 is based on [2, Chapter 5, p. 11], where we have added the corresponding broadcast times. It can be seen in Figure 3.2 that the second inequality of Theorem 3.8 is matched by complete k-ary trees with $k = \mathcal{O}(1)$. For complete graphs, expanders and hypercubes, the second inequality is tight up to a factor of $\mathcal{O}(\log n)$.

Graph	$\mathsf{COV}(G)$	$\mathbf{E}[\,\mathsf{RBA}(G)\,]$	$(1-\lambda_2)^{-1}$
path/cycle	n^2 [24]	n (Thm. 2.5)	n^2 [2, Ch. 5, p. 11]
complete $\mathcal{O}(1)$-ary tree	$n \log^2 n$ [32, Cor. 9]	$\log n$ (Thm. 2.5)	n [2, Ch. 5, p. 11]
complete graph	$n \log n$ [24]	$\log n$ [28]	1
expander	$n \log n$ [6]	$\log n$ [29]	1
hypercube	$n \log n$ [1]	$\log n$ [16]	$\log n$ [24]
$\sqrt{n} \times \sqrt{n}$-torus	$n \log^2 n$ [32, Thm. 4]	$\sqrt{n}$ (Thm. 2.5)	n [24]
$K_{n/2} \times K_2$	$n \log n$	$\log n$ [29]	n
lollipop	n^3 [24]	n	n^2 [2, Ch. 5, p. 22]

Figure 2: Comparison of the asymptotic order of the cover time, broadcast time and spectral gap of various graph classes. Recall that by Theorem 2.1, $(1-\lambda_2)^{-1}$ captures the mixing time up to logarithmic factors.

Let us consider the graph $K_{n/2} \times K_2$. One can easily verify that $\mathsf{COV}(G) = \mathcal{O}(n \log n)$, $\mathbf{E}[\,\mathsf{RBA}(G)\,] = \mathcal{O}(\log n)$, but $(1-\lambda_2)^{-1} = \Omega(n)$ (and consequently $\mathsf{MIX}_{e^{-1}}(G) = \Omega(n)$). Comparing these values with the ones of the complete graph, we see that there are graphs with an optimal cover time and optimal broadcast time, but $(1-\lambda_2)^{-1}$ may vary between $\Theta(1)$ and $\Omega(n)$. Hence the upper bound on the cover time based on the broadcast time can be a polynomial factor smaller than the corresponding bound (Theorem 2.4) based on the spectral gap $1-\lambda_2$. On the other hand, the following remark shows that by using the broadcast time instead of the spectral gap, we never lose more than a $\log^2 n$ factor:

Remark 3.9. For any regular graph G, the second bound of Theorem 3.8 implies

$$\mathsf{COV}(G) = \mathcal{O}\left(\frac{1}{1-\lambda_2} \cdot n \log^3 n\right).$$

In addition, Theorem 3.8 implies directly the following well-known bounds.

(1) Since $\mathbf{E}[\,\mathsf{RBA}(G)\,] = \mathcal{O}(n)$ for regular graphs [12, Prop. 1], we obtain $\max_{u,v} \mathsf{C}(u,v) = \mathcal{O}(n^2)$ for regular graphs [2, Ch. 6, Cor. 9].
(2) For bounded degree graphs, $\mathbf{E}[\,\mathsf{RBA}(G)\,] = \mathcal{O}(\mathrm{diam}(G))$ (by Theorem 2.5) implies $\max_{u,v} \mathsf{C}(u,v) = \mathcal{O}(n\,\mathrm{diam}(G))$ [2, Ch. 6, Cor. 8].
(3) Since $\max_{u,v} \mathbf{E}[\,\mathsf{RBA}(u,v)\,] = \mathcal{O}(n)$ [16], we obtain $\max_{u,v} \mathsf{C}(u,v) = \mathcal{O}(n^3)$ [2, Ch. 6, Thm. 1].

Finally, we give an application of Theorem 3.8 to certain power law random graphs (such networks are used to model real world networks [9]).

Definition 3.10. Given an n-dimensional vector $\mathbf{d} = (d_1, d_2, \ldots, d_n)$, the *generalized random graph* $G(\mathbf{d})$ is constructed as follows. Each edge $\{i,j\}, 1 \leqslant i,j \leqslant n$ exists with prob. $\frac{d_i \cdot d_j}{\sum_{k=1}^{n} d_k}$, independently of all other edges.

Theorem 3.11 ([11])**.** *Let* $\mathbf{d}$ *be a vector such that for all* i, $d_i > \log^c n$, *where* $c > 2$ *is some constant, and the number of vertices with expected degree* d *is proportional to* $(d - \log^c n)^{-1}$. *Then,* $G(\mathbf{d})$ *satisfies* $\mathsf{RBA}_{n^{-1}}(G(\mathbf{d})) = \mathcal{O}(\log n)$ *with probability* $1 - o(1)$.

Since the number of edges satisfies $|E(G(\mathbf{d}))| = \mathcal{O}(n \log^c n)$ with probability $1 - o(1)$ [9], we obtain by combining the theorem above with Theorem 3.8:

Corollary 3.12. *For* $G(\mathbf{d})$ *as in Theorem 3.11 we have* $\mathsf{COV}(G(\mathbf{d})) = \mathcal{O}(n \log^2 n)$ *with probability* $1 - o(1)$.

4. Lower Bounds on $\mathcal{R}(G)$

4.1. Sparse Graphs

Definition 4.1. Given a graph $G = (V, E)$, a set $\Pi \subseteq E(G)$ is called a *cutset separating* $u \in V$ *from* $v \in V$ if every path from u to v includes an edge of Π.

Proposition 4.2 ([23, p. 59],[26])**.** *For* $\{\Pi_i\}_{i=1}^{k}, k \in \mathbb{N}$, *being disjoint cutsets separating* u *from* v, $\mathsf{R}(u,v) \geqslant \sum_{i=1}^{k} |\Pi_i|^{-1}$.

Zuckerman [32] proved that for any two vertices u, v on a tree, $\mathsf{H}(u,v) \geqslant \mathrm{dist}(u,v)^2$. Using Proposition 4.2, we obtain the following generalization (a similar, but less tight bound follows from a result of [7]).

Corollary 4.3. *For any* $u, v \in V$ *of any graph* G, $\mathsf{C}(u,v) \geqslant 2 \cdot \mathrm{dist}(u,v)^2$. *On the other hand, there are graphs* G *and* $u, v \in V$ *such that* $\mathsf{H}(u,v) = \Theta(\mathrm{dist}(u,v)) = o(\mathrm{dist}(u,v)^2)$.

We remark that Corollary 4.3 is exact for paths (cf. [24]). Combining Corollary 4.3 with the known bounds from Theorem 2.3 and Theorem 2.5 yields:

Proposition 4.4. *For any graph* G *with maximum degree* δ, $\mathcal{R}(G) = \Omega(\frac{\sqrt{n}}{\Delta} \cdot \sqrt{\log n})$.

As demonstrated by the $\sqrt{n} \times \sqrt{n}$-torus where $\mathsf{RBA}_{n^{-1}}(G) = \Theta(\sqrt{n})$ (by Theorem 2.5) and $\mathsf{COV}(G) = \Theta(n \log^2 n)$ [32], this bound is tight up to a factor of $\log^{3/2} n$ for bounded degree graphs. The next result improves over Proposition 4.4 for dense graphs.

Theorem 4.5. *For any graph G with $\Delta = \mathcal{O}(\delta)$, $\mathcal{R}(G) = \Omega(\frac{\sqrt{n}}{\sqrt{\delta}} \cdot \frac{1}{\log n})$.*

4.2. Dense Graphs

In this section we present results that are tailored for dense graphs, e.g., graphs with minimum degree $\Theta(n)$. Consider a random walk $X_0 = s, X_1, \dots$ on G starting from s. Denote the number of visits to u until time t as $W_t(s,u) := |\{0 \leqslant t' \leqslant t : \ X_{t'} = u\}|$.

Definition 4.6 ([20, 31])**.** Consider a graph $G = (V, E)$ and a random walk starting from $s \in V$. Let

$$\mathsf{BLA}(s) := \mathbf{E}\left[\min\left\{ t \in \mathbb{N} \ \middle| \ \forall u \in V : \ \frac{1}{2} \cdot t\pi(u) \leqslant W_t(s,u) \leqslant 2 \cdot t\pi(u) \right\} \right].$$

Then, the *blanket time* of G is defined as $\mathsf{BLA}(G) := \max_{s \in V} \mathsf{BLA}(s)$.

Theorem 4.7 ([20])**.** *For any graph $G = (V, E)$, $\mathsf{BLA}(G) = \mathcal{O}(\mathsf{COV}(G) \cdot (\log \log n)^2)$.*

We also require the following simple graph-theoretical lemma.

Lemma 4.8. *For every graph G, there is a 2-cover X of G with $|X| \leqslant \lceil \frac{n}{\delta} \rceil$, i. e., there is a set $X \subseteq V$ such that for all $v \in V$ there is an $x \in X$ with $\operatorname{dist}(x, v) \leqslant 2$.*

Interestingly, it is known that there are graphs with minimum degree $\frac{n}{2}$ for which every 1-cover (i. e., dominating set) is of size $\Theta(\log n)$ [4], while the lemma above shows that every such graph has a 2-cover of constant size. We now prove the main result of Section 4.

Theorem 4.9. *For any graph with $\Delta = \mathcal{O}(\delta)$, $\mathbf{E}[\,\mathsf{RBA}(G)\,] = \mathcal{O}(\frac{1}{\delta} \cdot \mathsf{BLA}(G) + \frac{n^2}{\delta^2} \cdot \log^2 n)$.*

The following corollary follows immediately from Theorem 4.9 and Theorem 4.7.

Corollary 4.10. *For any graph $G = (V, E)$ with $\Delta = \mathcal{O}(\delta)$ we have $\mathcal{R}(G) = \Omega(\frac{\delta^2}{n} \cdot \frac{1}{\log n})$.*

Combining Corollary 4.10 with Theorem 3.8 for graphs with minimum degree $\Theta(n)$, we see that the cover time equals the broadcast time multiplied by n up to logarithmic factors. It is worth mentioning that for graphs with $\delta \geqslant \lfloor \frac{n}{2} \rfloor$, Chandra et al. [8, Theorem 3.3] proved that $\mathsf{COV}(G) = \Theta(n \log n)$. As pointed out by the same authors, $\mathsf{COV}(G)$ may be between $n \log n$ and $\Theta(n^2)$ if $\delta < \lfloor \frac{n}{2} \rfloor$. Now, Corollary 4.10 provides a parameter (the broadcast time) that captures the cover time not only for $\delta \geqslant \lfloor \frac{n}{2} \rfloor$, but also for $\delta = \Omega(n)$.

Proof of Theorem 4.9. Let us briefly describe the main idea of the proof. We first show that for every vertex u there is a fixed (independent of a concrete execution of RBA) set of vertices $Y(u) \subseteq V$ of size at least $\delta/12$ such that u informs each vertex in $Y(u)$ within $\mathcal{O}((n/\delta) \cdot \log^2 n)$ steps with high probability. We then establish that if a vertex u informs v in $\mathcal{O}((n/\delta) \cdot \log^2 n)$ steps with high probability, then also v informs u in $\mathcal{O}((n/\delta) \cdot \log^2 n)$ steps with high probability. Using this fact and Lemma 4.8 we find that there is a partitioning of V into a constant number of partitions with the following property: once a vertex in such a partition becomes informed, the whole partition becomes informed within $\mathcal{O}((n/\delta) \cdot \log^2 n)$ steps. Finally, we use a coupling between the random walk and the broadcast algorithm to show that if the random walk covers the whole graph quickly, then the rumor will also be quickly propagated from one partition to the other partitions. The formal proof follows.

Lemma 4.11. *For each $u \in V$ there is a set $Y(u) \subseteq V$ (independent of the execution of* RBA*) of size at least $\delta/12$ such that for every $v \in Y(u)$, $\mathsf{RBA}_{n^{-4}}(u,v) \leqslant 16C_1 \frac{n}{\delta} \log^2 n$, where $C_1 > 0$ is some constant.*

Lemma 4.12. *For any two vertices u, v in a graph G with $\Delta = \mathcal{O}(\delta)$, $\mathsf{RBA}_{n^{-4}}(v,u) \leqslant C_2 \cdot (\mathsf{RBA}_{n^{-4}}(u,v) + \log n)$, where $C_2 > 0$ is some constant.*

Consider the undirected auxiliary graph $\widehat{G} = (\widehat{V}, \widehat{E})$ defined as follows: $\widehat{V} := V$ and $\{u,v\} \in \widehat{E}$ iff

$$\max\{\mathsf{RBA}_{n^{-4}}(u,v), \mathsf{RBA}_{n^{-4}}(v,u)\} \leqslant C_2 \cdot \left(16C_1 \frac{n}{\delta} \log^2 n + \log n\right).$$

By the two lemmas above, $\delta(\widehat{G}) \geqslant \delta/12$. Hence Lemma 4.8 implies the existence of a 2-cover $\{u_1, u_2, \ldots, u_k\}$, $k \leqslant \lceil n/\delta \rceil$, of $\widehat{G}$. Therefore, the sets $U_i := \{v \in \widehat{V} \mid \mathrm{dist}_{\widehat{G}}(v, u_i) \leqslant 2\}, 1 \leqslant i \leqslant k$ form a (possibly non-disjoint) partitioning of $\widehat{V}$. Take a disjoint partitioning $V_1, V_2, \ldots, V_k$ such that for every $1 \leqslant i \leqslant k$, $V_i \subseteq U_i$. Consider now the directed graph $G' := (V', E')$ with $V' := \{V_1, V_2, \ldots, V_k\}$ and

$$E' := \left\{ (V_i, V_j) \mid \exists u \in V_i,\ 1 \leqslant t \leqslant 4 \cdot \frac{\deg(u)}{2|E|} \cdot \mathsf{BLA}(G) :\ N_{t,u} \in V_j \right\},$$

where $N_{t,u} \in N(u)$ is the vertex to which the random walk moves after the t-th visit of u.

Claim 4.13. *Let $s \in V_i$. With prob. $1/2$, there is a path from V_i to every V_j in G'.*

Reconsider now the partitioning $V_1, V_2, \ldots, V_k$, $k \leqslant \lceil n/\delta \rceil$, of $\widehat{V} = V$. Let $\mathrm{part}(u)$ be the function which assigns a vertex u the index of its partition. Let $\mathcal{B}$ be the event that $\forall u \in V : V_{\mathrm{part}(u)} \subseteq I_{\mathsf{RBA}(s,u)+\mathcal{O}(\frac{n}{\delta}\log^2 n)}$ holds, i. e., for all $u \in V$ it holds that once u is informed, the partition $V_{\mathrm{part}(u)}$ becomes completely informed within further $\mathcal{O}\left(\frac{n}{\delta}\log^2 n\right)$ steps. Fix some arbitrary vertex $u \in V$ and consider another vertex $w \in V_{\mathrm{part}(u)}$. By definition of $\widehat{G}$ and Lemma 4.12, there is path of length at most 4 from u to w in $\widehat{G}$. Hence once u is informed, w becomes informed within the next $\mathcal{O}(\frac{n}{\delta}\log^2 n)$ steps with probability $1 - 4n^{-4}$. Applying the union bound over $u \in V$ and $w \in V_{\mathrm{part}(u)}$, we get $\mathbf{Pr}[\mathcal{B}] \geqslant 1 - 4n^{-2}$.

Claim 4.14. *Conditioned on the events $\mathcal{A}$ and $\mathcal{B}$, all vertices of G become informed after $\mathcal{O}\left(\frac{1}{\delta} \cdot \mathsf{BLA}(G) + \frac{n^2}{\delta^2} \cdot \log^2 n\right)$ steps.*

To finish the proof of the Theorem, we apply the union bound to get $\mathbf{Pr}[\mathcal{A} \wedge \mathcal{B}] \geqslant 1 - \frac{1}{2} - 4n^{-2}$. So, with probability larger than $1/3$, all vertices of G become informed after at most $\mathcal{O}(\frac{1}{\delta} \cdot \mathsf{BLA}(G) + \frac{n^2}{\delta^2} \cdot \log^2 n)$ steps. Thus for every $k \in \mathbb{N}$, we succeed after $\mathcal{O}(k \cdot (\frac{1}{\delta} \cdot \mathsf{BLA}(G) + \frac{n^2}{\delta^2} \cdot \log^2 n))$ steps with probability $1 - (2/3)^k$ and hence the expected broadcast time is $\mathcal{O}(\frac{1}{\delta} \cdot \mathsf{BLA}(G) + \frac{n^2}{\delta^2} \cdot \log^2 n)$. ∎

4.3. Discussion

We first complement the lower bounds on $\mathcal{R}(G)$ by some concrete graphs. By a construction based on Harary graphs [18] and the two-dim. torus we obtain the following.

Theorem 4.15. *For any $\sqrt{n} \leqslant d \leqslant n-1$, there is a d-regular graph G with $\mathcal{R}(G) = \mathcal{O}(d \cdot \log n)$. Moreover, for any $1 \leqslant d \leqslant \sqrt{n}$ there is a graph with minimum degree d and maximum degree $d+1$ such that $\mathcal{R}(G) = \mathcal{O}(\sqrt{n} \cdot \log^2 n)$.*

While for certain degrees, a small polynomial gap remains between the examples of Theorem 4.15 and the bounds of Theorem 4.5 and Theorem 4.9 (cf. Figure 1), the quotient between cover time and diameter is minimized up to logarithmic factors by these examples.

Proposition 4.16. *For any graph G with $\Delta = \mathcal{O}(\delta)$, $\frac{\mathsf{COV}(G)}{\operatorname{diam}(G)} = \Omega(\max\{\sqrt{n}, \delta\} \cdot \sqrt{\log n})$.*

So far, in all considered graphs with a (nearly) optimal cover times and high broadcast time, the latter was caused by a large diameter. Therefore, one could try to throw in the lower bounds on $\operatorname{diam}(G)$ and ask the following question: Does $\mathsf{COV}(G) = \mathcal{O}(\operatorname{polylog}(n) \cdot \max\{n \log n, \operatorname{diam}(G)^2\}) \Leftrightarrow \mathbf{E}[\,\mathsf{RBA}(G)\,] = \mathcal{O}(\operatorname{polylog}(n) \cdot \max\{\operatorname{diam}(G), \log n\})$ hold? The answer is that both directions can be refuted by counter-examples, even for graphs where minimum and maximum degree coincide (up to constant factors).

5. Conclusion

Inspired by the intuition of Chandra et al. [8] about the relationship between cover time of random walks and the runtime of randomized broadcast, we devised the first formal results relating both times. As our main result in Section 3, we proved that the cover time of any graph G is upper bounded by $\mathcal{O}(\frac{E}{\delta} \log n)$ times the broadcast time. This result is tight for many graphs (at least up to a factor of $\log n$) and gives an upper bound on the cover time that is at least as good (and in certain cases much tighter than) the previous bound based on the spectral gap [6]. Moreover, this result implies several classic bounds on the cover time and an almost optimal upper bound on the cover time of certain random graphs that are used to model real world networks. In Section 4 we derived lower bounds on the ratio between the cover time and broadcast time. Together with our upper bound of Section 3, we established a surprisingly strong correspondence between the cover time and broadcast time on dense graphs. This positive result was complemented by the construction of certain graphs to demonstrate that this strong correspondence cannot be extended to sparser graphs. Nevertheless, our lower and upper bounds show that the relationship between cover time and broadcast time is substantially stronger than the relationship between any of these parameters and the mixing time (or the closely related spectral gap). In particular, our findings provide evidence for the following hierarchy for regular graphs:

$$\text{low mixing time} \Rightarrow \text{low broadcast time} \Rightarrow \text{low cover time},$$

which extends the following known relations: low mixing time $\Rightarrow$ low cover time ([1, 6, 10]) and low mixing time $\Rightarrow$ low broadcast time ([5, 12, 29]).

References

[1] D. J. Aldous. On the Time Taken by Random Walks on Finite Groups to Visit Every State. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, pages 361–374, 1983.

[2] D. J. Aldous and J. A. Fill. *Reversible Markov Chains and Random Walks on Graphs.* (draft at http://www.stat.berkeley.edu/aldous/RWG/book.html), 2002.

[3] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász, and C. Rackoff. Random Walks, Universal Traversal Sequences, and the Complexity of Maze Problems. In *20th IEEE Symp. on Found. of Computer Science (FOCS'79)*, pages 218–223, 1979.

[4] N. Alon and J. Spencer. *The Probabilistic Method*. John Wiley & Sons, 2nd edition, 2000.
[5] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized Gossip Algorithms. *IEEE Transactions on Information Theory and IEEE/ACM Transactions on Networking*, 52(6):2508–2530, 2006.
[6] A. Broder and A. Karlin. Bounds on the cover time. *Journal of Theoretical Prob.*, 2(1):101–120, 1989.
[7] T. K. Carne. A transmutation formula for markov chains. *Bulletin des Sciences Mathematiques*, 2(4):399–405, 1985.
[8] A.K. Chandra, P. Raghavan, W.L. Ruzzo, R. Smolensky, and P. Tiwari. The Electrical Resistance of a Graph Captures its Commute and Cover Times. *Computational Complexity*, 6(4):312–340, 1997.
[9] F. Chung, L. Lu, and V. Vu. Eigenvalues of random power law graphs. *Annals of Combinatorics*, 7:21–33, 2003.
[10] C. Cooper and A.M. Frieze. The Cover Time of Random Regular Graphs. *SIAM Journal of Discrete Mathematics*, 18(4):728–740, 2005.
[11] R. Elsässer. On Randomized Broadcasting in Power Law Networks. In *20th International Symposium on Distributed Computing (DISC'06)*, pages 370–384, 2006.
[12] R. Elsässer and T. Sauerwald. Broadcasting vs. Mixing and Information Dissemination on Cayley Graphs. In *24th International Symposium on Theoretical Aspects of Computer Science (STACS'07)*, pages 163–174, 2007.
[13] U. Feige. A Tight Lower Bound for the Cover Time of Random Walks on Graphs. *Random Structures & Algorithms*, 6(4):433–438, 1995.
[14] U. Feige. A Tight Upper Bound for the Cover Time of Random Walks on Graphs. *Random Structures & Algorithms*, 6(1):51–54, 1995.
[15] U. Feige. Collecting Coupons on Trees, and the Cover Time of Random Walks. *Comp. Complexity*, 6(4):341–356, 1997.
[16] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Randomized Broadcast in Networks. *Random Structures & Algorithms*, 1(4):447–460, 1990.
[17] J.A. Fill and R. Pemantle. Percolation, first-passage percolation and covering times for richardson's model on the n-cube. *The Annals of Applied Probability*, 3:593–629, 1993.
[18] F. Harary. The Maximum Connectivity of a Graph. *Proceedings of the National Academy of Sciences of the United States of America*, 48(7):1142–1146, 1962.
[19] H.W. Hethcore. Mathematics of infectious diseases. *SIAM Review 42*, pages 599–653, 2000.
[20] J.D. Kahn, J.H. Kim, L. Lovász, and V.H. Vu. The cover time, the blanket time and the Matthews bound. In *41st IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 467–475, 2000.
[21] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized Rumor Spreading. In *41st IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 565–574, 2000.
[22] W.O. Kermack and A.G. McKendrick. Contributions to the mathematical theory of epidemics. *Proceedings of the Royal Society*, 115A:700–721, 1927.
[23] D.A. Levin, Y. Peres, and E.L. Wilmer. *Markov Chains and Mixing Times*. (draft at http://www.oberlin.edu/markov/), 2006.
[24] L. Lovász. Random walks on graphs: A survey. *Combinatorics, Paul Erdös is Eighty*, 2:1–46, 1993.
[25] R. Lyons, R. Pemantle, and Y. Peres. Resistance Bounds for First-Passage-Percolation and Maximum Flow. *Journal of Combinatorial Theory (Series A)*, 86(1):158–168, 1999.
[26] C.St.J.A. Nash-Williams. Random walk and electric currents in networks. *Proceedings of the Cambridge Philosophical Society*, 55(1):181–194, 1959.
[27] M. E. J. Newman. The spread of epidemic disease on networks. *Physical Review E 66, 016128*, 2002.
[28] B. Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.
[29] T. Sauerwald. On Mixing and Edge Expansion Properties in Randomized Broadcasting. In *18th International Symposium on Algorithms and Computation (ISAAC'07)*, pages 196–207, 2007.
[30] A. Sinclair. Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow. *Combinatorics, Probability & Computing*, 1:351–370, 1992.
[31] P. Winkler and D. Zuckerman. Multiple Cover Time. *Random Structures & Alg.*, 9(4):403–411, 1996.
[32] D. Zuckerman. A Technique for Lower Bounding the Cover Time. *SIAM Journal on Discrete Math.*, 5(1):81–87, 1992.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 385–396
www.stacs-conf.org

ECONOMICAL CACHING

MATTHIAS ENGLERT [1] AND HEIKO RÖGLIN [2] AND JACOB SPÖNEMANN [3] AND BERTHOLD VÖCKING [4]

[1] DIMAP and Department of Computer Science, University of Warwick
E-mail address: englert@dcs.warwick.ac.uk

[2] Department of Computer Science, Boston University
E-mail address: Heiko@Roeglin.org

[3] Institute of Transport Science, RWTH Aachen University
E-mail address: Jacob.Spoenemann@rwth-aachen.de

[4] Department of Computer Science, RWTH Aachen University
E-mail address: voecking@cs.rwth-aachen.de

Abstract. We study the management of buffers and storages in environments with unpredictably varying prices in a competitive analysis. In the economical caching problem, there is a storage with a certain capacity. For each time step, an online algorithm is given a price from the interval $[1, \alpha]$, a consumption, and possibly a buying limit. The online algorithm has to decide the amount to purchase from some commodity, knowing the parameter α but without knowing how the price evolves in the future. The algorithm can purchase at most the buying limit. If it purchases more than the current consumption, then the excess is stored in the storage; otherwise, the gap between consumption and purchase must be taken from the storage. The goal is to minimize the total cost. Interesting applications are, for example, stream caching on mobile devices with different classes of service, battery management in micro hybrid cars, and the efficient purchase of resources.

First we consider the simple but natural class of algorithms that can informally be described as memoryless. We show that these algorithms cannot achieve a competitive ratio below $\sqrt{\alpha}$. Then we present a more sophisticated deterministic algorithm achieving a competitive ratio of

$$\frac{1}{W\left(\frac{1-\alpha}{e\alpha}\right)+1} \in \left[\frac{\sqrt{\alpha}}{\sqrt{2}}, \frac{\sqrt{\alpha}+1}{\sqrt{2}}\right] \ ,$$

where W denotes the Lambert W function. We prove that this algorithm is optimal and that not even randomized online algorithms can achieve a better competitive ratio. On the other hand, we show how to achieve a constant competitive ratio if the storage capacity of the online algorithm exceeds the storage capacity of an optimal offline algorithm by a factor of $\log \alpha$.

1998 ACM Subject Classification: F.1.2, F.2.2.

Key words and phrases: Online Algorithms, Competitive Analysis, Storage Management.

M. Englert is supported by EPSRC grant EP/F043333/1 and DFG grant WE 2842/1. H. Röglin is supported by a fellowship of the German Academic Exchange Service (DAAD) and by DFG through German excellence cluster UMIC. J. Spönemann is supported by DFG Research Training Group 1298 (AlgoSyn). B. Vöcking is supported by DFG through German excellence cluster UMIC.

1. Introduction

In many environments in which resources with unpredictably varying prices are consumed over time, the effective utilization of a storage can decrease the cost significantly. Since decisions have to be made without knowing how the price evolves in the future, storage management can naturally be formulated as an online problem in such environments. In the *economical caching problem* each time step is characterized by a price, a consumption, and a buying limit. In every such time step, an online algorithm has to decide the amount to purchase from some commodity. The algorithm can purchase at most the buying limit. If it purchases more than the current consumption, the excess is stored in a storage of limited capacity; otherwise, the gap between consumption and purchase must be taken from the storage.

This kind of problem does not only arise when purchasing resources like oil or natural gas, but also in other interesting application contexts. Let us illustrate this by two examples, one from the area of mobile communication and one dealing with the energy management in cars. The first example is stream caching on mobile devices with different communication standards like GSM, UMTS, WLAN. Since the price for transmitting data varies between the different standards and since for moving devices it is often unclear which standard will be available in the near future, the problem of cheaply caching a stream can be formulated in our framework. The second example is battery management in micro hybrid cars. In addition to a conventional engine, these cars have an electric motor without driving power that allows the engine to be restarted quickly after it had been turned off during coasting, breaking, or waiting. The power for the electric motor is taken from a battery that must be recharged by the alternator during drive. Since the effectiveness of the conventional engine depends on the current driving situation, the question of when and by how much to recharge the battery can be formulated as an economical caching problem.

Let α denote an upper bound on the price in any step that is known to the online algorithm. Formally, an instance of the economical caching problem is a sequence $\sigma_1\sigma_2\ldots$ in which every step σ_i consists of a price $\beta_i \in [1,\alpha]$, a consumption $v_i \geq 0$, and a buying limit $\ell_i \geq v_i$. During step σ_i, the algorithm has to decide the amount $B_i \in [0,\ell_i]$ to purchase. This amount has to be chosen such that neither the storage load drops below zero nor the storage load exceeds the capacity of the storage, which we can assume to be 1 without loss of generality. Formally, if L_{i-1} denotes the storage load after step σ_{i-1}, then B_i must be chosen such that $L_{i-1} + B_i - v_i \in [0,1]$. The restriction $\ell_i \geq v_i$ is necessary because otherwise covering the consumption might not be possible at all. The *economical caching problem without buying limits* is the special case in which all buying limits are set to infinity.

1.1. Our Results

First we observe that the following simple algorithm achieves a competitive ratio of $\sqrt{\alpha}$ (this also follows as a special case from Theorem 2.7): In every step σ_i with price $\beta_i \leq \sqrt{\alpha}$ buy as much as possible while adhering to the buying limit and the storage capacity. In all other steps buy only as much as necessary to maintain a non-negative storage load.

This algorithm belongs to a more general natural class of algorithms, namely algorithms with *fixed buying functions*. Given an arbitrary buying function $f\colon [1,\alpha] \to [0,1]$, we can define the following algorithm: For every σ_i the amount to purchase is chosen such that the storage load after the step is as close as possible to $f(\beta_i)$ taking into account the buying

limit. For example, the buying function f of the simple algorithm satisfies $f(x) = 1$ for $x \leq \sqrt{\alpha}$ and $f(x) = 0$ for $x > \sqrt{\alpha}$. Informally, algorithms with fixed buying functions can be seen as memoryless and vice versa, in the sense that the action in each step does only depend on the characteristics of that step and the current storage load. However, formally this intuitive view is incorrect since, due to the continuous nature of the problem, an algorithm can encode arbitrary additional information into the storage load. One of our results is a lower bound showing that there is no buying function that gives a better competitive factor than $\sqrt{\alpha}$.

Our main result, however, shows that this is not the best possible competitive factor. We present a more sophisticated deterministic algorithm that achieves a competitive ratio of

$$r := \frac{1}{W\left(\frac{1-\alpha}{e\alpha}\right)+1} \in \left[\frac{\sqrt{\alpha}}{\sqrt{2}}, \frac{\sqrt{\alpha}+1}{\sqrt{2}}\right] \ ,$$

where W denotes the Lambert W function (i.e., the inverse of $f(x) = x \cdot e^x$). We complement this result by a matching lower bound for randomized algorithms, showing that our algorithm is optimal and that randomization does not help. Our lower bounds hold even for the problem without buying limits.

Finally, we consider resource augmentation for the economical caching problem. We show that, for every $z \in \mathbb{N}\backslash\{1\}$, there is a buying function algorithm achieving a competitive ratio of $\sqrt[z]{\alpha}$ against an optimal offline algorithm whose storage capacity is by a factor of $z-1$ smaller than the storage capacity of the online algorithm. In particular, this implies that we obtain a buying function algorithm that is e-competitive against an optimal offline algorithm whose storage capacity is by a factor of $\max\{\lceil \ln(\alpha) \rceil - 1, 1\}$ smaller than the storage capacity of the online algorithm.

1.2. Previous Work

Although the economical caching problem is, in our opinion, a very natural problem with applications from various areas, it seems to have not been studied before in a competitive analysis. However, the problem bears some similarities to the one-way-trading problem introduced by El-Yaniv et al. [4]. In this problem, a trader needs to exchange some initial amount of money in some currency (say, dollars) to some other currency (say, euros). In each step, the trader obtains the current exchange rate and has to decide how much dollars to exchange. However, she cannot exchange euros back to dollars. El-Yaniv et al. present a tight bound of $\Theta(\log \phi)$ on the competitive ratio achievable for the one-way-trading problem, where ϕ denotes the ratio of the worst possible exchange rate and the best possible exchange rate. Results on variations of one- and two-way-trading can also be found in the book by Borodin and El-Yaniv [1] and in a survey by El-Yaniv [3]. In the two-way-trading problem, the trader can buy and sell in both directions. A related problem is portfolio management, which has been extensively studied (see, e.g., [2, 5, 6]).

The special case of the economical caching problem in which consumption occurs only in the last step can be viewed as a one-way-trading problem in which the trader does not start with a fixed amount of dollars but has a fixed target amount of euros. From our proof it is easy to see that our algorithm for the economical caching problem is strictly r-competitive on sequences that are terminated by a step with consumption 1 and price α. Additionally, the sequences used in the lower bound also have the property that they are terminated by such a step. Altogether, this implies that our algorithm is also optimal

for the one-way-trading problem with a fixed target amount and yields a strict competitive ratio of r for that problem.

1.3. Extensions

We can further generalize the economical caching problem. Each step may be characterized by a consumption and a monotonically increasing *price function* $p_i(x)\colon [0,\alpha] \to \mathbb{R}^+$, with $p_i(\alpha) \geq v_i$. The price function has the following meaning: The algorithm can buy up to an amount of $p_i(x)$ at rate at most x. The problem with a single price β_i and a buying limit ℓ_i for each step σ_i is a special case with $p_i(x) = 0$ for $x < \beta_i$ and $p_i(x) = \ell_i$ for $x \geq \beta_i$.

Such price functions appear, for example, implicitly in the stock market. At any given time, all sell orders for a specific stock in the order book define one price function since for every given price x there is a certain number of shares available with an ask price of at most x.

All our results also hold for this more general model. An (online) algorithm can transform an instance for the general problem into an instance of the special problem on the fly: A step with consumption v_i and price function p_i is transformed into a series of steps as follows: First we determine the maximum rate we have to pay to satisfy the demand as $\beta := \inf\{x \mid p_i(x) \geq v_i\}$. Then we generate the following steps (the upper value indicates the price, the middle value the consumption, and the lower value the buying limit)

$$\begin{pmatrix} 1 \\ p_i(1) \\ p_i(1) \end{pmatrix} \begin{pmatrix} 1+\varepsilon \\ p_i(1+\varepsilon) - p_i(1) \\ p_i(1+\varepsilon) - p_i(1) \end{pmatrix} \cdots \begin{pmatrix} \beta-\varepsilon \\ p_i(\beta-\varepsilon) - p_i(\beta-2\varepsilon) \\ p_i(\beta-\varepsilon) - p_i(\beta-2\varepsilon) \end{pmatrix} \begin{pmatrix} \beta \\ p_i(\beta) - p_i(\beta-\varepsilon) \\ p_i(\beta) - p_i(\beta-\varepsilon) \end{pmatrix}$$

for a small ε with $(\beta-1)/\varepsilon \in \mathbb{N}$. Finally, we append the following steps for the remaining prices

$$\begin{pmatrix} \beta+\varepsilon \\ 0 \\ p_i(\beta+\varepsilon) - p_i(\beta) \end{pmatrix} \cdots \begin{pmatrix} \alpha-\varepsilon \\ 0 \\ p_i(\alpha-\varepsilon) - p_i(\alpha-2\varepsilon) \end{pmatrix} \begin{pmatrix} \alpha \\ 0 \\ p_i(\alpha) - p_i(\alpha-\varepsilon) \end{pmatrix}$$

for a small ε with $(\alpha-\beta)/\varepsilon \in \mathbb{N}$.

If ε is small, this transformation does not change the cost of an optimal offline algorithm significantly and hence, our upper bounds on the competitive ratios still hold.

2. Upper Bound

2.1. The Optimal Offline Algorithm

To describe an optimal offline algorithm it is useful to track the cost-profile of the storage contents. For this, we define a monotonically decreasing function $g(x)\colon [0,\alpha] \to [0,1]$ that is initialized with $g(x) := 1$ and changes with each step. In the following, we denote the function $g(x)$ after step σ_i by $g_i(x)$ and the initial function by $g_0(x) = 1$.

The intuition behind $g(x)$ is that, assuming the storage of the optimal offline algorithm is completely filled after step σ_i, a $1 - g(x)$ fraction of the commodity stored in the storage was bought at price x or better.

The change of $g(x)$ from step to step follows two basic rules:

(1) Consumption is satisfied as cheap as possible, i.e., what we remove from the storage is what we bought at the lowest price.
(2) If we have stored something that was bought at a larger than the current price, replace it with commodity bought at the current price. That is, we revoke the decision of the past to buy at the worse price in favor of buying at the current, better price.

Formalizing this yields the following definition

$$g_i(x) := \begin{cases} \min\{g_{i-1}(x) + v_i, 1\} & \text{if } x \le \beta_i, \\ \max\{g_{i-1}(x) + v_i - \ell_i, 0\} & \text{if } x > \beta_i. \end{cases}$$

Using this definition, we can characterize the cost of an optimal offline algorithm. As described above, consumption is satisfied at the best possible price. This gives rise to the cost incurred in step σ_i, namely

$$C_i := \int_0^{\beta_i} \max\{g_{i-1}(x) + v_i - 1, 0\}\, dx \ .$$

Based on these values, we can characterize the cost of an optimal offline algorithm.

Lemma 2.1. *The cost of an optimal offline algorithm is exactly* $\sum_i C_i$.

Due to space limitations, we omit the technical but straightforward proof of this lemma.

2.2. The Optimal Online Algorithm

Our optimal $r := (W\left(\frac{1-\alpha}{e\alpha}\right) + 1)^{-1}$-competitive algorithm is based on the functions $g_i(x)$ introduced in Section 2.1. Note that an online algorithm can compute $g_i(x)$ since the function is solely based on information from the current and past steps. Let the storage level of the online algorithm after step σ_i be denoted by L_i. The initial storage load is $L_0 = 0$. Our algorithm bears some similarity with the following "threat-based" policy for one-way trading defined in [4]: In every step, convert just enough dollars to ensure that the desired competitive ratio would be obtained if in all following steps the exchange rate were equal to the worst possible rate. Our algorithm for the economical caching problem can be described as follows: In every step, the algorithm buys just enough to ensure that it would be strictly r-competitive if after the current step only one more step with consumption 1 and price α occurred that terminated the sequence.

This algorithm can be made explicit as follows: For each step σ_i of the input sequence with price β_i, buying limit ℓ_i, and consumption $v_i \le \ell_i$, the algorithm buys $B_i := v_i + r \cdot \int_1^{\alpha/r} \frac{g_{i-1}(x)-g_i(x)}{\alpha-x}\, dx$ at rate β_i. The storage level after this step is $L_i = L_{i-1} + r \cdot \int_1^{\alpha/r} \frac{g_{i-1}(x)-g_i(x)}{\alpha-x}\, dx$.

Lemma 2.2. *The algorithm above is admissible, that is, it does not buy more than the buying limit and after every step the storage level lies between 0 and 1.*

Proof. In a step σ_i, the algorithm buys

$$B_i = v_i + r \cdot \int_1^{\alpha/r} \frac{g_{i-1}(x) - g_i(x)}{\alpha - x}\, dx \ ,$$

which can be written as

$$\begin{aligned}
&= v_i + r \cdot \int_1^{\beta_i} \frac{g_{i-1}(x) - \min\{g_{i-1}(x) + v_i, 1\}}{\alpha - x}\, dx \\
&\qquad + r \cdot \int_{\beta_i}^{\alpha/r} \frac{g_{i-1}(x) - \max\{g_{i-1}(x) + v_i - \ell_i, 0\}}{\alpha - x}\, dx \\
&\leq v_i + r \cdot \int_{\beta_i}^{\alpha/r} \frac{g_{i-1}(x) - \max\{g_{i-1}(x) + v_i - \ell_i, 0\}}{\alpha - x}\, dx \\
&\leq v_i + r \cdot \int_{\beta_i}^{\alpha/r} \frac{\ell_i - v_i}{\alpha - x}\, dx \leq v_i + r \cdot \int_1^{\alpha/r} \frac{\ell_i - v_i}{\alpha - x}\, dx = \ell_i \ ,
\end{aligned}$$

where the last equation follows from the following observation.

Observation 2.3. For our choice of r,

$$\int_1^{\alpha/r} \frac{1}{\alpha - x}\, dx = \ln(\alpha - 1) - \ln(\alpha - \alpha/r) = \ln\left(1 - \frac{1}{\alpha}\right) - \ln\left(1 - \frac{1}{r}\right) = \frac{1}{r} \ .$$

This observation follows easily from the identity $\ln(-W(x)) = \ln(-x) - W(x)$.

The storage level after step σ_i is

$$\begin{aligned}
L_i &= L_{i-1} + r \cdot \int_1^{\alpha/r} \frac{g_{i-1}(x) - g_i(x)}{\alpha - x}\, dx = L_0 + r \cdot \int_1^{\alpha/r} \frac{g_0(x) - g_i(x)}{\alpha - x}\, dx \\
&= r \cdot \int_1^{\alpha/r} \frac{1 - g_i(x)}{\alpha - x}\, dx = 1 - r \cdot \int_1^{\alpha/r} \frac{g_i(x)}{\alpha - x}\, dx \ ,
\end{aligned}$$

where we use Observation 2.3 to obtain the last equation. This storage level is obviously at most 1. On the other hand,

$$L_i = 1 - r \cdot \int_1^{\alpha/r} \frac{g_i(x)}{\alpha - x}\, dx \geq 1 - r \cdot \int_1^{\alpha/r} \frac{1}{\alpha - x}\, dx = 0 \ ,$$

where the last step follows again from Observation 2.3.

Finally, let us observe that B_i is non-negative. From the definition of g_i it follows that $g_i(x) \leq g_{i-1}(x) + v_i$ for every x. Hence,

$$B_i \geq v_i + r \cdot \int_1^{\alpha/r} \frac{-v_i}{\alpha - x}\, dx = 0 \ ,$$

where the last equality is due to Observation 2.3. ∎

Theorem 2.4. *The algorithm above is $r := (W\left(\frac{1-\alpha}{e\alpha}\right) + 1)^{-1}$-competitive.*

Proof. To prove the theorem we show that, on any sequence, the cost of the algorithm above is at most r times the cost of the optimal offline algorithm plus α. Since α is a constant, this proves the theorem.

We already characterized the cost of an optimal offline algorithm in Section 2.1. The next step in our proof is to bound the C_i's from below. By Lemma 2.1, this yields a lower bound on the cost of an optimal offline algorithm, which is necessary for proving the desired competitive ratio.

Lemma 2.5. *For every step σ_i with $\beta_i \le \alpha/r$,*

$$C_i + \int_0^{\alpha/r} (g_i(x) - g_{i-1}(x))\,dx = \beta_i \cdot v_i - \int_{\beta_i}^{\alpha/r} \min\{\ell_i - v_i, g_{i-1}(x)\}\,dx \ .$$

For every step σ_i with $\beta_i > \alpha/r$,

$$C_i + \int_0^{\alpha/r} (g_i(x) - g_{i-1}(x))\,dx \ge \frac{\alpha}{r} \cdot v_i \ .$$

The only remaining part in the proof is to bound the cost of our algorithm from above. For this, observe that the cost that our algorithm incurs in step σ_i is exactly $\beta_i \cdot B_i$.

Lemma 2.6. *For every step σ_i with $\beta_i \le \alpha/r$,*

$$\beta_i \cdot B_i + \alpha(L_{i-1} - L_i) \le r \left(\beta_i \cdot v_i - \int_{\beta_i}^{\alpha/r} \min\{\ell_i - v_i, g_{i-1}(x)\}\,dx \right) \ .$$

For every step σ_i with $\beta_i > \alpha/r$,

$$\beta_i \cdot B_i + \alpha(L_{i-1} - L_i) \le \alpha \cdot v_i \ .$$

Given the previous lemmas, whose proof will be contained in the full version of this paper, the proof of the theorem follows from elementary calculations: Due to Lemma 2.5 and 2.6,

$$\beta_i \cdot B_i + \alpha(L_{i-1} - L_i) \le r \left(C_i + \int_0^{\alpha/r} (g_i(x) - g_{i-1}(x))\,dx \right) \ ,$$

for every step σ_i. Summing over all steps yields

$$\sum_{i=1}^{n} (\beta_i \cdot B_i) - \alpha(L_n - L_0) \le r \left(\sum_{i=1}^{n} C_i + \int_0^{\alpha/r} (g_n(x) - g_0(x))\,dx \right) \le r \cdot \sum_{i=1}^{n} C_i \ .$$

This concludes the proof of the theorem since the cost of our online algorithm is exactly $\sum_{i=1}^{n}(\beta_i \cdot B_i)$, $\alpha(L_n - L_0) \le \alpha$ and, due to Lemma 2.1, $\sum_{i=1}^{n} C_i$ is equal to the cost of an optimal offline algorithm. ∎

2.3. Algorithm for Larger Storage Capacities

In this section we present a buying function algorithm with a storage capacity of $\lceil \log \alpha / \log c \rceil - 1$ that is c-competitive against an optimal offline algorithm with storage capacity 1. In particular, this implies that for every $z \in \mathbb{N} \setminus \{1\}$, we have an algorithm with storage capacity $z - 1$ that achieves a competitive ratio of $\sqrt[z]{\alpha}$.

Let L_i denote the storage load after step σ_i. Further, we define a buying function

$$B(x) := \max\{\lceil \log \alpha / \log c \rceil - \lfloor \log x / \log c \rfloor - 1, 0\} \ .$$

For each step σ_i of the input sequence with price β_i, buying limit ℓ_i, and consumption $v_i \le \ell_i$, the algorithm buys

$$B_i := \max\{\min\{B(\beta_i) - L_{i-1} + v_i, \ell_i\}, 0\} \ .$$

Hence, the storage load L_i after the i-th step is $L_{i-1} + B_i - v_i$. Again, we have to argue that the algorithm is admissible, i.e., that $0 \le L_i \le \lceil \log \alpha / \log c \rceil - 1$. For $i = 0$ this is obviously the case since $L_0 = 0$. For $i \ge 1$, we observe that

$$\begin{aligned} L_i &= L_{i-1} + B_i - v_i \\ &= L_{i-1} + \max\{\min\{B(\beta_i) - L_{i-1} + v_i, \ell_i\}, 0\} - v_i \\ &= \max\{\min\{B(\beta_i), \ell_i + L_{i-1} - v_i\}, L_{i-1} - v_i\} \ . \end{aligned}$$

Now, on the one hand,

$$\max\{\min\{B(\beta_i), \ell_i + L_{i-1} - v_i\}, L_{i-1} - v_i\} \ge \min\{B(\beta_i), \ell_i + L_{i-1} - v_i\} \ge 0$$

due to the induction hypothesis $L_{i-1} \ge 0$ and since $B(\beta_i) \ge 0$ and $\ell_i \ge v_i$. On the other hand,

$$\max\{\min\{B(\beta_i), \ell_i + L_{i-1} - v_i\}, L_{i-1} - v_i\} \le \max\{B(\beta_i), L_{i-1} - v_i\} \le \lceil \log \alpha / \log c \rceil - 1$$

due to the induction hypothesis $L_{i-1} \le \lceil \log \alpha / \log c \rceil - 1$ and since $B(\beta_i) \le \lceil \log \alpha / \log c \rceil - 1$.

Theorem 2.7. *The above algorithm is c-competitive.*

Proof. To prove the theorem, we use the same functions $g_i(x)$ as in Theorem 2.4. Again, we can characterize the cost of an optimal offline algorithm as $\sum_i C_i$.

In addition, we introduce functions $f_i(x) \colon [0, \alpha] \to [0, \lceil \log \alpha / \log c \rceil - 1]$ defined by $f_0(x) := 0$ and

$$f_i(x) := \begin{cases} \min\{f_{i-1}(x) + B_i, L_i\} = B_i + \min\{f_{i-1}(x), L_{i-1} - v_i\} & \text{if } x \le \beta_i, \\ f_{i-1}(x) & \text{if } x > \beta_i. \end{cases}$$

Clearly, the cost of the online algorithm is equal to $\sum_i \beta_i \cdot B_i$. However, for our proof, we characterize the cost in a different way that is similar to our characterization of the optimal cost. For this, define

$$D_i := \int_0^{\beta_i} \max\{f_{i-1}(x) - L_i + B_i, 0\} \, dx \ .$$

Lemma 2.8. *For every j,*

$$\sum_{i=1}^{j} \beta_i \cdot B_i = \int_0^{\alpha} f_j(x) \, dx + \sum_{i=1}^{j} D_i \ .$$

The goal is to relate D_i to C_i in order to prove the theorem. More precisely, we show that, for every i, $D_i \le c \cdot C_i$. This yields the theorem as

$$\begin{aligned} \sum_{i=1}^{j} \beta_i \cdot B_i &= \int_0^{\alpha} f_j(x) + \sum_{i=1}^{j} D_i \\ &\le \alpha \cdot (\lceil \log \alpha / \log c \rceil - 1) + \sum_{i=1}^{j} D_i \\ &\le \alpha \cdot (\lceil \log \alpha / \log c \rceil - 1) + c \cdot \sum_{i=1}^{j} C_i \ . \end{aligned}$$

In order to show $D_i \le c \cdot C_i$, we need the following invariant.

Lemma 2.9. *For every i and $x \in [0, \alpha/c]$, $L_i - f_i(c \cdot x) - 1 + g_i(x) \geq 0$.*

Using this lemma we obtain

$$\begin{aligned} D_i &= \int_0^{\beta_i} \max\{f_{i-1}(x) - L_i + B_i, 0\}\, dx = \int_0^{\beta_i} \max\{f_{i-1}(x) - L_{i-1} + v_i, 0\}\, dx \\ &\leq \int_0^{\beta_i} \max\{g_{i-1}(x/c) - 1 + v_i, 0\}\, dx = c \cdot \int_0^{\beta_i/c} \max\{g_{i-1}(x) - 1 + v_i, 0\}\, dx \\ &\leq c \cdot \int_0^{\beta_i} \max\{g_{i-1}(x) + v_i - 1, 0\}\, dx = c \cdot C_i \ . \end{aligned}$$

The proofs of Lemmas 2.8 and 2.9 will be contained in the full version of this paper. ■

3. Lower Bounds

3.1. General Lower Bound

Theorem 3.1. *The competitive ratio of any randomized online algorithm for the economical caching problem is at least*

$$r := \frac{1}{W\left(\frac{1-\alpha}{e\alpha}\right) + 1} \ .$$

This also holds for the economical caching problem without buying limits.

Proof. Let A denote an arbitrary randomized online algorithm. For every $\beta \in [1, \alpha/r]$, we construct a sequence Σ_β. This sequence starts with a series Σ'_β of steps without a buying limit, without consumption, and with prices decreasing from α/r to β. To be more precise, let the prices in this series of steps be

$$\frac{\alpha}{r}, \frac{\alpha}{r} - \varepsilon, \frac{\alpha}{r} - 2\varepsilon, \ldots, \beta + \varepsilon, \beta,$$

for a small $\varepsilon > 0$ with $(\alpha/r - \beta)/\varepsilon \in \mathbb{N}$. Since we can choose the discretization parameter ε arbitrarily small, we assume in the following that the prices decrease continuously from α/r to β, to avoid the cumbersome notation caused by discretization. Finally, the sequence Σ_β is obtained by appending one step without a buying limit, consumption 1, and price α to Σ'_β.

Due to the last step with consumption 1 and price α, we can assume that after a sequence of the form Σ_β algorithm A has an empty storage. Otherwise, we can easily modify A such that this property is satisfied without deteriorating its performance. Given this assumption, the behavior of algorithm A on sequences Σ_β can be completely described in terms of a monotonically decreasing *buying function* $f\colon [1, \alpha/r] \to [0, 1]$ with the following meaning: after the subsequence Σ'_β with decreasing prices from α/r to β, the expected storage level of A is $f(\beta)$. Using linearity of expectation, the expected costs of A on Σ_β can be expressed as

$$C_A(\Sigma_\beta) = C_f(\beta) = (1 - f(\beta)) \cdot \alpha + \beta \cdot f(\beta) + \int_\beta^{\alpha/r} f(x)\, dx \ .$$

The first term results from the fact that in the last step of Σ_β algorithm A has to purchase the amount of $1 - f(\beta)$ for price α. The remaining term is illustrated in Figure 1.

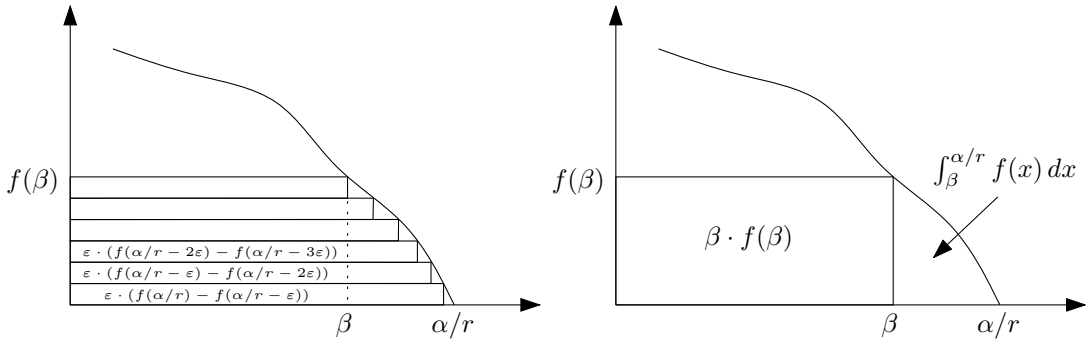


Figure 1: The first figure illustrates the cost of algorithm A on the discrete sequence, and the second figure illustrates its cost on the continuous sequence. Since the function f is monotone, it is integrable on the compact set $[\beta, \alpha/r]$, which in turn implies that for $\varepsilon \to 0$ the costs on the discrete and continuous sequence coincide.

In addition to the actual buying function f of algorithm A, we also consider the buying function g defined by

$$g(x) = r \cdot \left(\ln\left(1 - \frac{x}{\alpha}\right) - \ln\left(1 - \frac{1}{r}\right) \right) \ .$$

This buying function has the property that for all $\beta \in [1, \alpha/r]$

$$C_g(\beta) = (1 - g(\beta)) \cdot \alpha + \beta \cdot g(\beta) + \int_\beta^{\alpha/r} g(x)\,dx = r \cdot \beta \ ,$$

as shown by the following calculation:

$$\begin{aligned}
&(1 - g(\beta))\alpha + \beta \cdot g(\beta) + \int_\beta^{\alpha/r} g(x)\,dx \\
&= (1 - g(\beta))\alpha + \beta \cdot g(\beta) + \int_\beta^{\alpha/r} r \cdot \left(\ln\left(1 - \frac{x}{\alpha}\right) - \ln\left(1 - \frac{1}{r}\right) \right) dx \\
&= (1 - g(\beta))\alpha + \beta \cdot g(\beta) + \left[r \cdot (-\alpha + x) \left(\ln\left(1 - \frac{x}{\alpha}\right) - 1 \right) \right]_\beta^{\alpha/r} - r \cdot \left(\frac{\alpha}{r} - \beta \right) \ln\left(1 - \frac{1}{r}\right) \\
&= (1 - g(\beta))\alpha + \beta \cdot g(\beta) + r \cdot \left((-\alpha + \beta) \ln\left(1 - \frac{1}{r}\right) - (-\alpha + \beta) \ln\left(1 - \frac{\beta}{\alpha}\right) + \left(\beta - \frac{\alpha}{r}\right) \right) \\
&= (1 - g(\beta))\alpha + \beta \cdot g(\beta) + (\alpha - \beta) \cdot g(\beta) + r \cdot \left(\beta - \frac{\alpha}{r}\right) = r \cdot \beta \ .
\end{aligned}$$

Furthermore, g is a valid buying function as it is monotonically decreasing, $g(\alpha/r) = 0$, and

$$g(1) = r \cdot \left(\ln\left(1 - \frac{1}{\alpha}\right) - \ln\left(1 - \frac{1}{r}\right) \right) = 1 \ ,$$

which follows from Observation 2.3.

In order to show the lower bound on A's competitive ratio, we distinguish between two cases: either $f(x) > g(x)$ for all $x \in [1, \alpha/r]$ or there exists an $x \in [1, \alpha/r]$ with $f(x) \leq g(x)$. In the former case, we set $\beta = 1$ and, according to the previous calculations, obtain that $C_A(\Sigma_1) = C_f(1) > C_g(1) = r$. Since the cost of an optimal offline algorithm on Σ_1 is 1, the competitive ratio of algorithm A is bounded from below by r in this case. Now let us consider the case that there exists an $x \in [1, \alpha/r]$ with $f(x) \leq g(x)$. In this case, we set

$$\beta = \sup\{x \in [1, \alpha/r] \mid f(x) \leq g(x)\} \ .$$

Since $f(x) \geq g(x)$ for all $x \geq \beta$, we obtain

$$C_A(\Sigma_\beta) = C_f(\beta) \geq C_g(\beta) = r\beta \ .$$

Combining this with the observation that the cost of an optimal offline algorithm on the sequence Σ_β is β implies that, also in this case, the competitive ratio of A is bounded from below by r.

The argument above shows only that no algorithm can be strictly r'-competitive for $r' < r$ (in fact, it is easy to see that no algorithm can be strictly r'-competitive for $r' < \alpha$). However, the assumption that A has an empty storage after each sequence Σ_β allows us to repeat an arbitrary number of sequences of this kind without affecting the argumentation above, showing that no algorithm can be better than r-competitive. Observe that the buying function of algorithm A can be different in each repetition, which, however, cannot help to obtain a better competitive ratio because β is adopted appropriately in each repetition. ∎

3.2. Lower Bound for Algorithms with Fixed Buying Functions

Theorem 3.2. *The competitive ratio of any randomized online algorithm for the economical caching problem with a fixed buying function is at least $\sqrt{\alpha}$. This also holds for the economical caching problem without buying limits.*

Proof. Let us first consider an algorithm A with an arbitrary but monotonically decreasing buying function f. We will later argue how to extend the proof to functions that are not necessarily monotonically decreasing. We construct a sequence Σ on which A is at least $\sqrt{\alpha}$-competitive as follows: Σ starts with a sequence Σ' that is similar to Σ'_1 from the proof of Theorem 3.1 with the only exception that we decrease the efficiency from α to 1. To be precise, in every step in this sequence there is no consumption, no buying limit, and the prices are

$$\alpha, \alpha - \varepsilon, \alpha - 2\varepsilon, \ldots, 1 + \varepsilon, 1 \ ,$$

for a small ε with $(\alpha - 1)/\varepsilon \in \mathbb{N}$. As in the proof of Theorem 3.1, we simplify the notation by assuming that the price decreases continuously from α to 1. The cost of A on this sequence is

$$q := 1 + \int_1^\alpha f(x)\, dx \ .$$

Let us assume that $f(1) = 1$. Due to the construction of the sequence Σ this can only reduce the cost of A on Σ. We can also assume that $f(\alpha) = 0$ because if A purchases anything at price α, it can easily be seen that A cannot be better than α-competitive.

Now we distinguish between two cases: if $q \geq \sqrt{\alpha}$, then the sequence Σ is formed by appending one step with price α, consumption 1, and no buying limit to Σ'. The cost of an optimal offline algorithm on this sequence is 1, whereas the cost of A is q. Hence, in this case, algorithm A is at least $\sqrt{\alpha}$-competitive.

Now let us assume that $q \leq \sqrt{\alpha}$. After the sequence Σ', the price increases again from 1 to α but this time with consumption. There still is no buying limit and the prices and consumptions are as follows (the upper value indicates the price, the lower value the consumption):

$$\binom{1+\varepsilon}{f(1)-f(1+\varepsilon)}\binom{1+2\varepsilon}{f(1+\varepsilon)-f(1+2\varepsilon)}\cdots\binom{\alpha-\varepsilon}{f(\alpha-2\varepsilon)-f(\alpha-\varepsilon)}\binom{\alpha}{f(\alpha-\varepsilon)} \ .$$

Let us call this sequence Σ''. Observe that consumptions and prices are chosen such that A does not purchase anything during the sequence Σ''. The sequence Σ is formed by appending one step with price α, consumption 1, and no buying limit to $\Sigma'\Sigma''$. On this sequence, the optimal cost is $1+q$: The optimal offline algorithm purchases an amount of 1 in the last step of Σ' for price 1, and then it purchases in every step of Σ'' exactly the consumption. This way the storage is completely filled after the sequence Σ'' and no further cost is incurred in the final step. Similar arguments as in the proof of Theorem 3.1 show that the cost during the sequence Σ'' is q. Since algorithm A does not purchase anything during Σ'', it has to purchase an amount of 1 for the price of α in the final step. Hence, its total cost is $q+\alpha$. For $q \leq \sqrt{\alpha}$, we have

$$\frac{q+\alpha}{q+1} \geq \frac{\sqrt{\alpha}+\alpha}{\sqrt{\alpha}+1} = \sqrt{\alpha} \ .$$

Since $f(\alpha) = 0$, algorithm A has an empty storage after this sequence. Hence, we can repeat this sequence an arbitrary number of times, proving the theorem.

If the buying function f is not monotonically decreasing, we can, for the purpose of this proof, replace f by the monotonically decreasing function $f^*(x) := \sup\{f(y) \mid y \geq x\}$. An algorithm with buying function f behaves the same as an algorithm with buying function f^* on the sequences constructed in this lower bound with respect to f^*. ∎

References

[1] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[2] Thomas M. Cover and Erik Ordentlich. Universal portfolios with side information. *IEEE Transactions on Information Theory*, 42(2):348–363, 1996.

[3] Ran El-Yaniv. Competitive solutions for online financial problems. *ACM Comput. Surv.*, 30(1):28–69, 1998.

[4] Ran El-Yaniv, Amos Fiat, Richard M. Karp, and G. Turpin. Optimal search and one-way trading online algorithms. *Algorithmica*, 30(1):101–139, 2001.

[5] David P. Helmbold, Robert E. Schapire, Yoram Singer, and Manfred K. Warmuth. On-line portfolio selection using multiplicative updates. In *ICML*, pages 243–251, 1996.

[6] Erik Ordentlich and Thomas M. Cover. On-line portfolio selection. In *COLT*, pages 310–313, 1996.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 397–408
www.stacs-conf.org

COMPUTING GRAPH ROOTS WITHOUT SHORT CYCLES

BABAK FARZAD [1] AND LAP CHI LAU [2] AND VAN BANG LE [3] AND NGUYEN NGOC TUY† [3,4]

[1] Department of Mathematics, Brock University, Canada.
E-mail address: bfarzad@brocku.ca

[2] Department of Computer Science and Engineering, The Chinese University of Hong Kong.
E-mail address: chi@cse.cuhk.edu.hk

[3] Universität Rostock, Institut für Informatik, Germany.
E-mail address: {le,nn024}@informatik.uni-rostock.de

[4] Hong Duc University, Vietnam.
E-mail address: nntuy@yahoo.com

Abstract. Graph G is the square of graph H if two vertices x, y have an edge in G if and only if x, y are of distance at most two in H. Given H it is easy to compute its square H^2, however Motwani and Sudan proved that it is NP-complete to determine if a given graph G is the square of some graph H (of girth 3). In this paper we consider the characterization and recognition problems of graphs that are squares of graphs of small girth, i.e. to determine if $G = H^2$ for some graph H of small girth. The main results are the following.

- There is a graph theoretical characterization for graphs that are squares of some graph of girth at least 7. A corollary is that if a graph G has a square root H of girth at least 7 then H is unique up to isomorphism.
- There is a polynomial time algorithm to recognize if $G = H^2$ for some graph H of girth at least 6.
- It is NP-complete to recognize if $G = H^2$ for some graph H of girth 4.

These results almost provide a dichotomy theorem for the complexity of the recognition problem in terms of girth of the square roots. The algorithmic and graph theoretical results generalize previous results on tree square roots, and provide polynomial time algorithms to compute a graph square root of small girth if it exists. Some open questions and conjectures will also be discussed.

Key words and phrases: Graph roots, Graph powers, Recognition algorithms, NP-completeness.

† This author is supported by the Ministry of Education and Training, Vietnam, Grant No. 3766/QD-BGD & DT.

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 397-408
http://drops.dagstuhl.de/opus/volltexte/2009/1827

1. Introduction

Root and *root finding* are concepts familiar to most branches of mathematics. In graph theory, H is a *square root* of G and G is the *square* of H if two vertices x, y have an edge in G if and only if x, y are of distance at most two in H. Graph square is a basic operation with a number of results about its properties in the literature. In this paper we are interested in the characterization and recognition problems of graph squares. Ross and Harary [22] characterized squares of trees and showed that tree square roots, when they exist, are unique up to isomorphism. Mukhopadhyay [20] provided a characterization of graphs which have a square root, but this is not a good characterization in the sense that it does not give a short certificate when a graph does not have a square root. In fact, such a good characterization may not exist as Motwani and Sudan proved that it is NP-complete to determine if a given graph has a square root [19]. On the other hand, there are polynomial time algorithms to compute the tree square root [17, 14, 15, 3, 4], a bipartite graph square root [15], and a proper interval graph square root [16].

The algorithms for computing tree square roots and bipartite graph square roots are based on the fact that the square roots have no cycles and no odd cycles respectively. Since computing the graph square uses only local information from the first and the second neighborhood, it is plausible that there are polynomial time algorithms to compute square roots that have no short cycles (locally tree-like), and more generally to compute square roots that have no short odd cycles (locally bipartite). The *girth* of a graph is the length of a shortest cycle. In this paper we consider the characterization and recognition problems of graphs that are squares of graphs of small girth, i.e. to determine if $G = H^2$ for some graph H of small girth.

The main results of this paper are the following. In Section 2 we will provide a good characterization for graphs that are squares of some graph of girth at least 7. This characterization not only leads to a simple algorithm to compute a square root of girth at least 7 but also shows such a square root, if it exists, is unique up to isomorphism. Then, in Section 3, we will present a polynomial time algorithm to compute a square root of girth at least 6, or report that none exists. In Section 4 we will show that it is NP-complete to determine if a graph G has a square root of girth 4. Finally, we discuss some open questions and conjectures.

These results almost provide a dichotomy theorem for the complexity of the recognition problem in terms of girth of the square roots. The algorithmic and graph theoretical results considerably generalize previous results on tree square roots. We believe that our algorithms can be extended to compute square roots with no short odd cycles (locally bipartite), and in fact one part of the algorithm for computing square roots of girth at least 6 uses only the assumption that the square roots have no 3 cycles or 5 cycles. Coloring properties of squares in terms of girth of the roots have been considered in the literature [2, 5, 11]; our algorithms would allow those results to apply even though a square root was not known apriori.

Definitions and notation: All graphs considered are finite, undirected and simple. Let $G = (V_G, E_G)$ be a graph. We often write $xy \in E_G$ for $\{x, y\} \in E_G$. Following [19, 16], we sometimes also write $x \leftrightarrow y$ for the adjacency of x and y in the graph in question; this is particularly the case when we describe reductions in NP-completeness proofs.

The *neighborhood* $N_G(v)$ in G of a vertex v is the set all vertices in G adjacent to v and the *closed neighborhood* of v in G is $N_G[v] = N_G(v) \cup \{v\}$. Set $\deg_G(v) = |N_G(v)|$, the

degree of v in G. We call vertices of degree one in G *end-vertices* of G. A *center vertex* of G is one that is adjacent to all other vertices.

Let $d_G(x, y)$ be the length, i.e., number of edges, of a shortest path in G between x and y. Let $G^k = (V_G, E^k)$ with $xy \in E^k$ if and only if $1 \leq d_G(x, y) \leq k$ denote the *k-th power of G*. If $G = H^k$ then G is the k-th power of the graph H and H is a *k-th root* of G. Since the power of a graph H is the union of the powers of the connected components of H, we may assume that all graphs considered are connected.

A set of vertices $Q \subseteq V_G$ is called a *clique* in G if every two distinct vertices in Q are adjacent; a *maximal clique* is a clique that is not properly contained in another clique. A *stable set* is a set of pairwise non-adjacent vertices. Given a set of vertices $X \subseteq V_G$, the subgraph induced by X is written $G[X]$ and $G - X$ stands for $G[V \setminus X]$. If $X = \{a, b, c, \ldots\}$, we write $G[a, b, c, \ldots]$ for $G[X]$. Also, we often identify a subset of vertices with the subgraph induced by that subset, and vice versa.

The *girth* of G, $\mathrm{girth}(G)$, is the smallest length of a cycle in G; in case G has no cycles, we set $\mathrm{girth}(G) = \infty$. In other words, G has girth k if and only if G contains a cycle of length k but does not contain any (induced) cycle of length $\ell = 3, \ldots, k-1$. Note that the girth of a graph can be computed in $O(nm)$ time, where n and m are the number of vertices, respectively, edges of the input graph [13].

A complete graph is one in which every two distinct vertices are adjacent; a complete graph on k vertices is also denoted by K_k. A *star* is a graph with *at least two* vertices that has a center vertex and the other vertices are pairwise non-adjacent. Note that a star contains at least one edge and at least one center vertex; the center vertex is unique whenever the star has more than two vertices.

2. Squares of graphs with girth at least seven

In this section, we give a good characterization of graphs that are squares of a graph of girth at least seven. Our characterization leads to a simple polynomial-time recognition for such graphs.

Proposition 2.1. *Let G be a connected, non-complete graph such that $G = H^2$ for some graph H.*
(i) If $\mathrm{girth}(H) \geq 6$ and v is a vertex with $\deg_H(v) \geq 2$ then $N_H[v]$ is a maximal clique in G;
(ii) If $\mathrm{girth}(H) \geq 7$ and Q is a maximal clique in G then $Q = N_H[v]$ for some vertex v where $\deg_H(v) \geq 2$.

Proof. (i) Let v be a vertex with $\deg_H(v) \geq 2$. Clearly, $Q = N_H[v]$ is a clique in G. Consider an arbitrary vertex w outside Q; in particular, w is non-adjacent in H to v. If w is non-adjacent in H to all vertices in Q, then $d_H(w, v) > 2$. If w is adjacent in H to a vertex $x \in Q - v$, let $y \in Q \setminus \{v, x\}$. Then $N_H[w] \cap N_H[y] = \emptyset$ (otherwise H would contain a cycle of length at most five), hence $d_H(w, y) > 2$. Thus, in any case, w cannot be adjacent, in G, to all vertices in Q, and so Q is a maximal clique in G.

(ii) Let Q be a maximal clique in G and $v \in Q$ be a vertex that maximizes $|Q \cap N_H[v]|$. We prove that $Q = N_H[v]$. It can be seen that by the maximality of Q, $\deg_H(v) \geq 2$. Now, we show that if $w \in Q \setminus N_H[v]$ and $x \in Q \cap N_H[v]$, then $wx \notin E_H$: As $w \notin N_H[v]$, this is clear in case $x = v$. So, let $x \neq v$ and assume to the contrary that $wx \in E_H$. Then, by the choice of v, there exists a vertex $w' \in Q \setminus N_H[x]$, $w' \in N_H[v]$. Note that $w'x, w'w \notin E_H$

because H has no C_3, C_4. As $ww' \in E_G \setminus E_H$, there exists a vertex $u \notin \{w, w', x, v\}$ with $uw, uw' \in E_H$. But then $H[w, w', x, v, u]$ contains a C_4 or C_5. Contradiction.

Finally, we show that $Q \subseteq N_H[v]$, and so, by the maximality of Q, $Q = N_H[v]$: Assume otherwise and let $w \in Q \setminus N_H[v]$. As $wv \in E_G \setminus E_H$, there exists a vertex x such that $xw, xv \in E_H$, and so, $x \in N_H[v] \setminus Q$. By the maximality of Q, x must be non-adjacent (in G) to a vertex $w' \in Q$. In fact, $w' \in Q \setminus N_H[v]$ as x is adjacent in G to every vertex in $N_H[v]$. Since $w'v \in E_G \setminus E_H$, there exists a vertex a such that $aw', av \in E_H$; note that $a \notin \{x, w\}$. Now, if $ww' \in E_H$ then $H[w, w', a, v, x]$ contains a cycle of length at most five. If $ww' \notin E_H$, let b be a vertex such that $bw, bw' \in E_H$; possibly $b = a$. Then $H[w, w', a, b, v, x]$ contains a cycle of length at most six. In any case we have a contradiction, hence $Q \setminus N_H[v] = \emptyset$. ∎

The 5-cycle C_5 and the 6-cycle C_6 show that (i), respectively, (ii) in Proposition 2.1 is best possible with respect to the girth condition of the root. More generally, the maximal cliques in the square of the subdivision of any complete graph on $n \geq 3$ vertices do not satisfy Condition (ii).

Definition 2.2. Let G be an arbitrary graph. An edge of G is called *forced* if it is contained in (at least) two distinct maximal cliques in G.

Proposition 2.3. *Let G be a connected, non-complete graph such that $G = H^2$ for some graph H with girth at least seven, and let F be the subgraph of G consisting of all forced edges of G. Then*
(i) F is obtained from H by deleting all end-vertices in H;
(ii) for every maximal clique Q in G, $F[Q \cap V_F]$ is a star; and
(iii) every vertex in $V_G - V_F$ belongs to exactly one maximal clique in G.

Proof. First we observe that xy is a forced edge in G iff xy is an edge in H with $\deg_H(x) \geq 2$ and $\deg_H(y) \geq 2$. Now, (i) follows directly from the above observations. For (ii), consider a maximal clique Q in G. By Proposition 2.1, $Q = N_H[v]$ for some vertex v with $\deg_H(v) \geq 2$. Let X be the set of all neighbors of v in H that are end-vertices in H and $Y = N_H(v) \setminus X$. Since G is not complete, $Y \neq \emptyset$. By (i), $X \cap V_F = \emptyset$, hence $F[Q \cap V_F] = F[\{v\} \cup Y]$ which implies (ii). For (iii), consider a vertex $u \in V_G - V_F$ and a maximal clique Q containing u. Then, u cannot belong to Y and therefore Q is the only maximal clique containing u. ∎

We now are able to characterize squares of graphs with girth at least seven as follows.

Theorem 2.4. *Let G be a connected, non-complete graph. Let F be the subgraph of G consisting of all forced edges in G. Then G is the square of a graph with girth at least seven if and only if the following conditions hold.*
(i) Every vertex in $V_G - V_F$ belongs to exactly one maximal clique in G.
(ii) Every edge in F belongs to exactly two distinct maximal cliques in G.
(iii) Every two non-disjoint edges in F belong to a common maximal clique in G.
(iv) For each maximal clique Q of G, $F[Q \cap V_F]$ is a star.
(v) F is connected and has girth at least seven.

Proof. For the only if-part, (ii) and (iii) follow easily from Proposition 2.1, and (i), (iv) and (v) follow directly from Proposition 2.3.

For the if-part, let G be a connected graph satisfying (i) – (v). We will construct a spanning subgraph H of G with girth at least seven such that $G = H^2$ as follows. For each edge xy in F let, by (ii) and (iv), $Q \neq Q'$ be *the* two maximal cliques in G with $Q \cap Q' = \{x, y\}$. Let, without loss of generality, $|Q \cap V_F| \geq |Q' \cap V_F|$. Assuming x is

a center vertex of the star $F[Q \cap V_F]$, then y is a center vertex of the star $F[Q' \cap V_F]$: Otherwise, by (iv), x is the center vertex of the star $F[Q' \cap V_F]$ and there exists some $y' \in Q' \cap V_F$ such that $yy' \notin F$; note that $xy' \in F$ (by (iv)). As $|Q \cap V_F| \geq |Q' \cap V_F|$, there is an edge $xz \in F - xy$ in $Q - Q'$. By (iii), $zy' \in E_G$. Now, as Q' is maximal, the maximal clique Q'' containing x, y, z, y' is different from Q'. But then $\{y, y'\} \subseteq Q' \cap Q''$, i.e., $yy' \in F$, hence F contains a triangle xyy', contradicting (v).

Thus, assuming x is a center vertex of the star $F[Q \cap V_F]$, y is a center vertex of the star $F[Q' \cap V_F]$. Then put the edges xq, $q \in Q - x$, and yq', $q' \in Q' - y$, into H.

By construction, $F \subseteq H \subseteq G$ and by (i),

$$\text{for all vertices } u \in V_H \setminus V_F,\ \deg_H(u) = 1, \tag{2.1}$$

$$\forall v \in V_F, \forall a, b \in V_H \text{ with } va, vb \in E_H :\ a \text{ and } b \text{ belong to the same clique in } G. \tag{2.2}$$

Furthermore, as every maximal clique in G contains a forced edge (by (iv)), H is a spanning subgraph of G. Moreover, F is an induced subgraph of H: Consider an edge $xy \in E_H$ with $x, y \in V_F$. By construction of H, x or y is a center vertex of the star $F[Q \cap V_F]$ for some maximal clique Q in G. Since $x, y \in V_F$, xy must be an edge of this star, i.e., $xy \in E_F$. Thus, F is an induced subgraph of H. In particular, by (2.1) and (v), H is connected and $\text{girth}(H) = \text{girth}(F) \geq 7$.

Now, we complete the proof of Theorem 2.4 by showing that $G = H^2$. Let $uv \in E_G \setminus E_H$ and let Q be a maximal clique in G containing uv. By (iv), Q contains a forced edge xy and x or y is a center vertex of the star $F[Q \cap V_F]$. By construction of H, xu and xv, or else yu and yv are edges of H, hence $uv \in E_{H^2}$. This proves $E_G \subseteq E_{H^2}$. Now, let $ab \in E_{H^2} \setminus E_H$. Then there exists a vertex x such that $xa, xb \in E_H$. By (2.1), $x \in V_F$, and by (2.2), $ab \in E_G$. This proves $E_{H^2} \subseteq E_G$. ∎

Corollary 2.5. *Given a graph $G = (V_G, E_G)$, it can be recognized in $O(|V_G|^2 \cdot |E_G|)$ time if G is the square of a graph H with girth at least seven. Moreover, such a square root, if any, can be computed in the same time.*

Proof. Note that by Proposition 2.1, any square of an n-vertex graph with girth at least seven has at most n maximal cliques. Now, to avoid triviality, assume G is connected and non-complete. We first use the algorithm in [23] to list the maximal cliques in G in time $O(n^2 m)$. If there are more than n maximal cliques, G is not the square of any graph with girth at least seven. Otherwise, compute the forced edges of G to form the subgraph F of G. This can be done in time $O(n^2)$ in an obvious way. Conditions (i) – (v) in Theorem 2.4 then can be tested within the same time bound, as well as the square root H, in case all conditions are satisfied, according to the proof of Theorem 2.4. ∎

Corollary 2.6. *The square roots with girth at least seven of squares of graphs with girth at least seven are unique, up to isomorphism.*

Proof. Let G be the square of some graph H with girth ≥ 7. If G is complete, clearly, every square root with girth ≥ 6 of G must be isomorphic to the star $K_{1,n-1}$ where n is the vertex number of G.

Thus, let G be non-complete, and let F be the subgraph of G formed by the forced edges. If F has only one edge, G clearly consists of exactly two maximal cliques, Q_1, Q_2, say, and $Q_1 \cap Q_2$ is the only forced edge of G. Then, it is easily seen that every square root with girth ≥ 6 of G must be isomorphic to the double star T having center edge $v_1 v_2$ and $\deg_T(v_i) = |Q_i|$.

So, assume F has at least two edges. Then for each two maximal cliques Q, Q' in G with $Q \cap Q' = \{x, y\}$, x or y is the unique center vertex of the star $F[V_F \cap Q]$ or $F[V_F \cap Q']$. Hence, for any end-vertex u of H, i.e., $u \in V_G - V_F$, the neighbor of u in F is unique. Since F is the graph resulting from H by deleting all end-vertices, H is therefore unique. ∎

2.1. Further Considerations

Squares of bipartite graphs can be recognized in $O(\Delta \cdot M(n))$ time in [15], where $\Delta = \Delta(G)$ is the maximum degree of the n-vertex input graph G and $M(n)$ is the time needed to perform the multiplication of two $n \times n$-matrices. However, no good characterization is known so far. As bipartite graphs with girth at least seven are exactly the (C_4, C_6)-free bipartite graphs, we immediately have:

Corollary 2.7. *Let G be a connected, non-complete graph. Let F be the subgraph of G consisting of all forced edges in G. Then G is the square of a (C_4, C_6)-free bipartite graph if and only if the following conditions hold.*
(i) Every vertex in $V_G - V_F$ belongs to exactly one maximal clique in G.
(ii) Every edge in F belongs to exactly two distinct maximal cliques in G.
(iii) Every two non-disjoint edges in F belong to the same maximal clique in G.
(iv) For each maximal clique Q of G, $F[Q \cap V_F]$ is a star.
(v) F is a connected (C_4, C_6)-free bipartite graph.
Moreover, squares of (C_4, C_6)-free bipartite graphs can be recognized in $O(n^2m)$ time, and the (C_4, C_6)-free square bipartite roots of such squares are unique, up to isomorphism.

Using the results in this section, we obtain a new characterization for tree squares that allow us to derive the known results on tree square roots easily.

It was shown in [17] that CLIQUE and STABLE SET remain NP-complete on squares of graphs (of girth three). Another consequence of our results is.

Corollary 2.8. *The weighted version of* CLIQUE *can be solved in $O(n^2m)$ time on squares of graphs with girth at least 7, where n and m are the number of vertices, respectively, edges of the input graph.*

Proof. Let $G = (V_G, E_G)$ be the square of some graph with girth at least seven. By Proposition 2.1, G has $O(|V_G|)$ maximal cliques. By [23], all maximal cliques in G then can be listed in time $O(|V_G| \cdot |E_G| \cdot |V_G|)$. ∎

In [12], it was shown that STABLE SET is even NP-complete on squares of the subdivision of some graph (i.e. the squares of the total graph of some graph). As the subdivision of a graph has girth at least six, STABLE SET therefore is NP-complete on squares of graphs with girth at least six.

3. Squares of graphs with girth at least six

In this section we will show that squares of graphs with girth at least six can be recognized efficiently. Formally, we will show that the following problem

SQUARE OF GRAPH WITH GIRTH AT LEAST SIX

Instance: A graph G.
Question: Does there exist a graph H with girth at least 6 such that $G = H^2$?

is polynomially solvable (Theorem 3.5).

Similar to the algorithm in [15], our recognition algorithm consists of two steps. The first step (subsection 3.1) is to show that if we fix a vertex $v \in V$ and a subset $U \subseteq N_G(v)$, then there is at most one $\{C_3, C_5\}$-free (locally bipartite) square root graph H of G with $N_H(v) = U$. Then, in the second step (subsection 3.2), we show that if we fix an edge $e = uv \in E_G$, then there are at most two possibilities of $N_H(v)$ for a square root H with girth at least 6. Furthermore, both steps can be implemented efficiently, and thus it will imply that SQUARE OF GRAPH WITH GIRTH AT LEAST SIX is polynomially solvable.

3.1. Square root with a specified neighborhood

This subsection deals with the first auxiliary problem.

$\{C_3, C_5\}$-FREE SQUARE ROOT WITH A SPECIFIED NEIGHBORHOOD

Instance: A graph G, $v \in V_G$ and $U \subseteq N_G(v)$.
Question: Does there exist a $\{C_3, C_5\}$-free graph H such that $H^2 = G$ and $N_H(v) = U$?

An efficient recognition algorithm for $\{C_3, C_5\}$-FREE SQUARE ROOT WITH A SPECIFIED NEIGHBORHOOD relies on the following fact.

Lemma 3.1. *Let $G = H^2$ for some $\{C_3, C_5\}$-free graph H. Then, for all vertices $x \in V$ and all vertices $y \in N_H(x)$, $N_H(y) = N_G(y) \cap \big(N_G[x] \setminus N_H(x)\big)$.*

Proof. First, consider an arbitrary vertex $w \in N_H(y) - x$. Clearly, $w \in N_G(y)$, as well $w \in N_G(x)$. Also, since H is C_3-free, $wx \notin E_H$. Thus $w \in N_G(y) \cap \big(N_G(x) \setminus N_H(x)\big)$.

Conversely, let w be an arbitrary vertex in $N_G(y) \cap \big(N_G[x] \setminus N_H(x)\big)$. Assuming $wy \notin E_H$, then $w \neq x$ and there exist vertices z and z' such that $zx, zw \in E_H$ and $z'y, z'w \in E_H$. As H is C_3-free, $zy \notin E_H$, $z'x \notin E_H$, and $zz' \notin E_H$. But then x, y, w, z and z' induce a C_5 in H, a contradiction. Thus $w \in N_H(y)$. ∎

Recall that $M(n)$ stands for the time needed to perform a matrix multiplication of two $n \times n$ matrices; currently, $M(n) = O(n^{2.376})$.

Theorem 3.2. $\{C_3, C_5\}$-FREE SQUARE ROOT WITH A SPECIFIED NEIGHBORHOOD *has at most one solution. The unique solution, if any, can be constructed in time $O(M(n))$.*

Proof. Given G, $v \in V_G$ and $U \subseteq N_G(v)$, assume H is a $\{C_3, C_5\}$-free square root of G such that $N_H(v) = U$. Then, by Lemma 3.1, the neighborhood in H of each vertex $u \in U$ is uniquely determined by $N_H(u) = N_G(u) \cap \big(N_G[v] \setminus U\big)$. By repeatedly applying Lemma 3.1 for each $v' \in U$ and $U' = N_H(v')$ and noting that all considered graphs are connected, we can conclude that H is unique.

Lemma 3.1 also suggests the following BFS-like procedure, Algorithm 1 below, for constructing the $\{C_3, C_5\}$-free square root H of G with $U = N_H(v)$, if any.

It can be seen, by construction, that H is $\{C_3, C_5\}$-free, and thus the correctness of Algorithm 1 follows from Lemma 3.1. Moreover, since every vertex is enqueued at most once, lines 1–13 take $O(m)$ steps, $m = |E_G|$. Checking if $G = H^2$ (line 14) takes $O(M(n))$ steps, $n = |V_G|$. ∎

ALGORITHM 1

Input: A graph G, a vertex $v \in V_G$ and a subset $U \subseteq N_G(v)$.
Output: A $\{C_3, C_5\}$-free graph H with $H^2 = G$ and $N_H(v) = U$, or else 'NO' if such a square root H of G does not exist.

```
1.  Add all edges vu, u ∈ U, to E_H
2.  Q ← ∅
3.  for each u ∈ U do
4.      enqueue(Q, u)
5.      parent(u) ← v
6.  while Q ≠ ∅ do
7.      u ← dequeue(Q)
8.      set W := N_G(u) ∩ (N_G(parent(u)) \ N_H(parent(u)))
9.      for each w ∈ W do
10.         add uw to E_H
11.         if parent(w) = ∅
12.         then parent(w) ← u
13.              enqueue(Q, w)
14. if G = H^2 then return H
15.    else return 'NO'
```

3.2. Square root with a specified edge

This subsection discusses the second auxiliary problem.

GIRTH ≥ 6 ROOT GRAPH WITH ONE SPECIFIED EDGE

Instance: A graph G and an edge $xy \in E_G$.
Question: Does there exist a graph H with girth at least six such that $H^2 = G$ and $xy \in E_H$?

The question is easy if $|G| \leq 2$. So, for the rest of this section, assume that $|G| > 2$. Then, we will reduce this problem to $\{C_3, C_5\}$-FREE SQUARE ROOT WITH A SPECIFIED NEIGHBORHOOD. Given a graph G and an edge xy of G, write $C_{xy} = N_G(x) \cap N_G(y)$, i.e., C_{xy} is the set of common neighbors of x and y in G.

Lemma 3.3. *Suppose H is of girth at least 6, $xy \in E_H$ and $H^2 = G$. Then $G[C_{xy}]$ has at most two connected components. Moreover, if A and B are the connected components of $G[C_{xy}]$ (one of them maybe empty) then (i) $A = N_H(x) - y$ and $B = N_H(y) - x$, or (ii) $B = N_H(x) - y$ and $A = N_H(y) - x$.*

By Lemma 3.3, we can solve GIRTH ≥ 6 ROOT GRAPH WITH ONE SPECIFIED EDGE as follows: Compute C_{xy}. If $G[C_{xy}]$ has more than two connected components, there is no solution. If $G[C_{xy}]$ is connected, solve $\{C_3, C_5\}$-FREE SQUARE ROOT WITH A SPECIFIED NEIGHBORHOOD for inputs $I_1 = (G, v = x, U = C_{xy} + y)$ and $I_2 = (G, v = y, U = C_{xy} + x)$. If, for I_1 or I_2, Algorithm 1 outputs H and H is C_4-free, then H is a solution. In other cases there is no solution. If $G[C_{xy}]$ has two connected components, A and B, solve $\{C_3, C_5\}$-FREE SQUARE ROOT WITH A SPECIFIED NEIGHBORHOOD for inputs $I_1 = (G, v = x, U = A + y)$, $I_2 = (G, v = x, U = B + y)$, $I_3 = (G, v = y, U = A + x)$, $I_4 = (G, v = y, U = B + x)$, and

make a decision similar as before. In this way, checking if a graph is C_4-free is the most expensive step, and we obtain

Theorem 3.4. GIRTH ≥ 6 ROOT GRAPH WITH ONE SPECIFIED EDGE *can be solved in time* $O(n^4)$.

Let $\delta = \delta(G)$ denote the minimum vertex degree in G. Now we can state the main result of this section as follows.

Theorem 3.5. SQUARE OF GRAPH WITH GIRTH AT LEAST SIX *can be solved in time* $O(\delta \cdot n^4)$.

Proof. Given G, let x be a vertex of minimum degree in G. For each vertex $y \in N_G(x)$ check if the instance $(G, xy \in E_G)$ for GIRTH ≥ 6 ROOT GRAPH WITH ONE SPECIFIED EDGE has a solution. ∎

4. Squares of graphs with girth four

Note that the reductions for proving the NP-completeness results by Motwani and Sudan [19] show that recognizing squares of graphs with girth three is NP-complete. In this section we show that the following problem is NP-complete.

SQUARE OF GRAPH WITH GIRTH FOUR

Instance: A graph G.
Question: Does there exist a graph H with girth 4 such that $G = H^2$?

Observe that SQUARE OF GRAPH WITH GIRTH FOUR is in NP. We will reduce the following NP-complete problem SET SPLITTING [8, Problem SP4], also known as HYPERGRAPH 2-COLORABILITY, to it.

SET SPLITTING

Instance: Collection D of subsets of a finite set S.
Question: Is there a partition of S into two disjoint subsets S_1 and S_2 such that each subset in D intersects both S_1 and S_2?

Our reduction is a modification of the reductions for proving the NP-completeness of SQUARE OF CHORDAL GRAPH [16, Theorem 3.5] and for CUBE OF BIPARTITE GRAPH [15, Theorem 7.6]. We also apply the tail structure of a vertex v, first described in [19], to ensure that v has the same neighbors in any square root H of G.

Lemma 4.1 ([19])**.** *Let a, b, c be vertices of a graph G such that (i) the only neighbors of a are b and c, (ii) the only neighbors of b are a, c, and d, and (iii) c and d are adjacent. Then the neighbors, in $V_G - \{a, b, c\}$, of d in any square root of G are the same as the neighbors, in $V_G - \{a, b, d\}$, of c in G; see Figure* 1.

We now are going to describe the reduction. Let $S = \{u_1, \ldots, u_n\}$, $D = \{d_1, \ldots, d_m\}$ where $d_j \subseteq S$, $1 \leq j \leq m$, be an instance of SET SPLITTING. We construct an instance $G = G(D, S)$ for SQUARE OF GRAPH WITH GIRTH FOUR as follows.

The vertex set of graph G consists of:
(I) U_i, $1 \leq i \leq n$. Each 'element vertex' U_i corresponds to the element u_i in S.
(II) D_j, $1 \leq j \leq m$. Each 'subset vertex' D_j corresponds to the subset d_j in D.
(III) D_j^1, D_j^2, D_j^3, $1 \leq j \leq m$. Each three 'tail vertices' D_j^1, D_j^2, D_j^3 of the subset vertex D_j correspond to the subset d_j in D.
(IV) S_1, S_1', S_2, S_2', four 'partition vertices'.
(V) X, a 'connection vertex'.

The edge set of graph G consists of:
(I) Edges of tail vertices of subset vertices:
For all $1 \leq j \leq m$: $D_j^3 \leftrightarrow D_j^2$, $D_j^3 \leftrightarrow D_j^1$, $D_j^2 \leftrightarrow D_j^1$, $D_j^2 \leftrightarrow D_j$, $D_j^1 \leftrightarrow D_j$, and for all i, $D_j^1 \leftrightarrow U_i$ whenever $u_i \in d_j$.
(II) Edges of subset vertices:
For all $1 \leq j \leq m$: $D_j \leftrightarrow S_1$, $D_j \leftrightarrow S_1'$, $D_j \leftrightarrow S_2$, $D_j \leftrightarrow S_2'$, $D_j \leftrightarrow X$, $D_j \leftrightarrow U_i$ for all i, and $D_j \leftrightarrow D_k$ for all k with $d_j \cap d_k \neq \emptyset$.
(III) Edges of element vertices:
For all $1 \leq i \leq n$: $U_i \leftrightarrow X$, $U_i \leftrightarrow S_1$, $U_i \leftrightarrow S_2$, $U_i \leftrightarrow S_1'$, $U_i \leftrightarrow S_2'$, and $U_i \leftrightarrow U_{i'}$ for all $i' \neq i$.
(IV) Edges of partition vertices:
$S_1 \leftrightarrow X$, $S_1 \leftrightarrow S_1'$, $S_1 \leftrightarrow S_2'$, $S_2 \leftrightarrow X$, $S_2 \leftrightarrow S_1'$, $S_2 \leftrightarrow S_2'$, $S_1' \leftrightarrow X$, $S_2' \leftrightarrow X$.

Clearly, G can be constructed from D, S in polynomial time. For an illustration, given $S = \{u_1, u_2, u_3, u_4, u_5\}$ and $D = \{d_1, d_2, d_3, d_4\}$ with $d_1 = \{u_1, u_2, u_3\}$, $d_2 = \{u_2, u_5\}$, $d_3 = \{u_3, u_4\}$, and $d_4 = \{u_1, u_4\}$, the graph G is depicted in Figure 2. In the figure, the two dotted lines from a vertex to the clique $\{U_1, U_2, U_3, U_4, U_5, X\}$ mean that the vertex is adjacent to all vertices in that clique.

Note that, apart from the three vertices X, S_1', and S_2' (or, symmetrically, X, S_1, and S_2), our construction is the same as those in [16, §3.1.1]. While S_1 and S_2 will represent a partition of the ground set S (Lemma 4.3), the vertices X, S_1', and S_2' allow us to make a square root of G being C_3-free (Lemma 4.2).

Lemma 4.2. *If there exists a partition of S into two disjoint subsets S_1 and S_2 such that each subset in D intersects both S_1 and S_2, then there exists a graph H with girth four such that $G = H^2$.*

In the above example, $S_1 = \{u_1, u_3, u_5\}$ and $S_2 = \{u_2, u_4\}$ is a possible legal partition of S. The corresponding graph H constructed in the proof of Lemma 4.2 is depicted in Figure 3.

Lemma 4.3. *If H is a square root of G, then there exists a partition of S into two disjoint subsets S_1 and S_2 such that each subset in D intersects both S_1 and S_2.*

Note that in Lemma 4.3 above we did not require that H has girth four. Thus, any square root of G–particularly, any square root with girth four–will tell us how to do set splitting. Together with Lemma 4.2 we conclude:

Theorem 4.4. SQUARE OF GRAPH WITH GIRTH FOUR *is NP-complete.*

5. Conclusion and open problems

We have shown that squares of graphs with girth at least six can be recognized in polynomial time. We have found a good characterization for squares of graphs with girth at least seven that gives a faster recognition algorithm in this case. For squares of graphs with girth at most four we have shown that recognizing the squares of such graphs is NP-complete.

The complexity status of computing square root with girth (exactly) five is not yet determined. However, we believe that this problem should be efficiently solvable. Also, we believe that the algorithm to compute a square root of girth 6 can be extended to compute

a square root with no C_3 or C_5. More generally, let k be a positive integer and consider the following problem.

k-POWER OF GRAPH WITH GIRTH $\geq 3k-1$

Instance: A graph G.
Question: Does there exist a graph H with girth $\geq 3k-1$ such that $G = H^k$?

Conjecture 5.1. k-POWER OF GRAPH WITH GIRTH $\geq 3k-1$ is polynomially solvable.

The truth of the above conjecture together with the results in this paper would imply a complete dichotomy theorem: SQUARES OF GRAPHS OF GIRTH g is polynomial if $g \geq 5$ and NP-complete otherwise.

References

[1] Geir Agnarsson, Raymond Greenlaw, Magnús M. Halldórsson, On powers of chordal graphs and their colorings, *Congressus Numer.* 144 (2000) 41–65.
[2] Noga Alon, Bojan Mohar, The chromatic number of graph powers, *Combinatorics, Probability and Computing* 11 (2002) 1–10.
[3] Andreas Brandstädt, Van Bang Le, and R. Sritharan, Structure and linear time recognition of 4-leaf powers, *ACM Transactions on Algorithms*, to appear.
[4] Maw-Shang Chang, Ming-Tat Ko, and Hsueh-I Lu, Linear time algorithms for tree root problems, *Lecture Notes in Computer Science*, 4059 (2006) 411–422.
[5] Daniel W. Cranston, Seog-Jin Kim, List-coloring the square of a subcubic graph, *J. Graph Theory* 57 (2007) 65–87.
[6] Elias Dahlhaus, P. Duchet, On strongly chordal graphs, *Ars Combin.* 24 B (1987) 23–30.
[7] F. Escalante, L. Montejano, and T. Rojano, Characterization of n-path graphs and of graphs having nth root, *J. Combin. Theory B* 16 (1974) 282–289.
[8] Michael R. Garey, David S. Johnson, *Computers and Intractability–A Guide to the Theory of NP-Completeness*, Freeman, New York (1979), twenty-third printing 2002.
[9] Martin C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York (1980).
[10] Frank Harary, R.M. Karp, and W.T. Tutte, A criterion for planarity of the square of a graph, *J. Combin. Theory* 2 (1967) 395–405.
[11] Frédéric Havet, Choosability of the square of planar subcubic graphs with large girth, *Discrete Math.*, to appear.
[12] J.D. Horton, K. Kilakos, Minimum edge dominating sets, *SIAM J. Discrete Math.* 6 (1993) 375–387.
[13] Alon Itai, Michael Rodeh, Finding a minimum circuit in a graph, *SIAM J. Computing* 7 (1978) 413–423.
[14] Paul E. Kearney, Derek G. Corneil, Tree powers, *J. Algorithms* 29 (1998) 111–131.
[15] Lap Chi Lau, Bipartite roots of graphs, *ACM Transactions on Algorithms* 2 (2006) 178–208.
[16] Lap Chi Lau, Derek G. Corneil, Recognizing powers of proper interval, split and chordal graphs, *SIAM J. Discrete Math.* 18 (2004) 83–102.
[17] Yaw.-Ling Lin, Steven S. Skiena, Algorithms for square roots of graphs, *SIAM J. Discrete Math.* 8 (1995) 99–118.
[18] A. Lubiw, Γ-free matrices, *Master Thesis*, Dept. of Combinatorics and Optimization, University of Waterloo, Canada, 1982.
[19] Rajeev Motwani, Madhu Sudan, Computing roots of graphs is hard, *Discrete Appl. Math.* 54 (1994) 81-88.
[20] A. Mukhopadhyay, The square root of a graph, *J. Combin. Theory* 2 (1967) 290-295.
[21] A. Raychaudhuri, On powers of strongly chordal and circular arc graphs, *Ars Combin.* 34 (1992) 147–160.
[22] I.C. Ross, Frank Harary, The square of a tree, *Bell System Tech. J.* 39 (1960) 641–647.
[23] Shuji Tsukiyama, Mikio Ide, Hiromu Ariyoshi, and Isao Shirakawa, A new algorithm for generating all the maximal independent sets, *SIAM J. Computing* 6 (1977) 505–517.

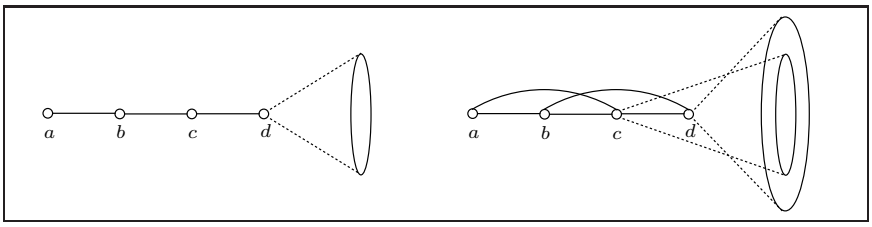


Figure 1: Tail in H (left) and in $G = H^2$ (right)

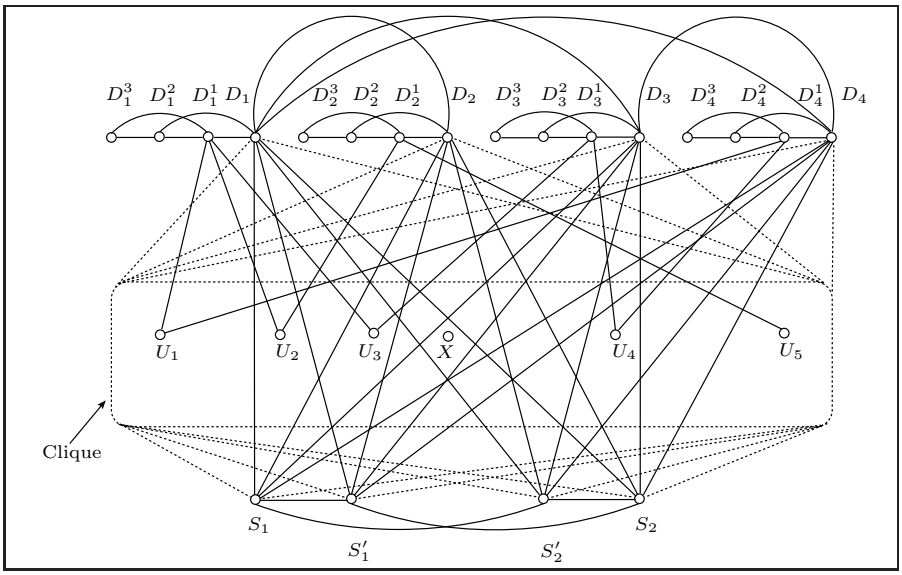


Figure 2: An example of G

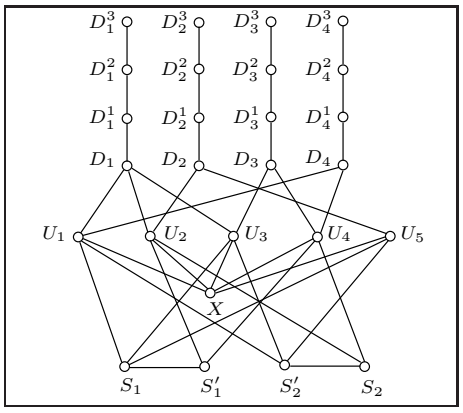


Figure 3: An example of root H with girth 4

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 409–420
www.stacs-conf.org

A GENERALIZATION OF NEMHAUSER AND TROTTER'S LOCAL OPTIMIZATION THEOREM

MICHAEL R. FELLOWS [1] AND JIONG GUO [2] AND HANNES MOSER [2] AND ROLF NIEDERMEIER [2]

[1] PC Research Unit, Office of DVC (Research),
University of Newcastle, Callaghan, NSW 2308, Australia.
E-mail address: michael.fellows@newcastle.edu.au

[2] Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany.
E-mail address: (guo,moser,niedermr)@minet.uni-jena.de

Abstract. The Nemhauser-Trotter local optimization theorem applies to the NP-hard Vertex Cover problem and has applications in approximation as well as parameterized algorithmics. We present a framework that generalizes Nemhauser and Trotter's result to vertex deletion and graph packing problems, introducing novel algorithmic strategies based on purely combinatorial arguments (not referring to linear programming as the Nemhauser-Trotter result originally did).

We exhibit our framework using a generalization of Vertex Cover, called Bounded-Degree Deletion, that has promise to become an important tool in the analysis of gene and other biological networks. For some fixed $d \geq 0$, Bounded-Degree Deletion asks to delete as few vertices as possible from a graph in order to transform it into a graph with maximum vertex degree at most d. Vertex Cover is the special case of $d = 0$. Our generalization of the Nemhauser-Trotter theorem implies that Bounded-Degree Deletion has a problem kernel with a linear number of vertices for every constant d. We also outline an application of our extremal combinatorial approach to the problem of packing stars with a bounded number of leaves. Finally, charting the border between (parameterized) tractability and intractability for Bounded-Degree Deletion, we provide a W[2]-hardness result for Bounded-Degree Deletion in case of unbounded d-values.

1998 ACM Subject Classification: F.2.2, G.2.1, G.2.2, G.2.3.

Key words and phrases: Algorithms, computational complexity, NP-hard problems, W[2]-completeness, graph problems, combinatorial optimization, fixed-parameter tractability, kernelization.

The first author was supported by the Australian Research Council. Work done while staying in Jena as a recipient of the Humboldt Research Award of the Alexander von Humboldt Foundation, Bonn, Germany. The second author was supported by the DFG, Emmy Noether research group PIAF, NI 369/4, and project DARE, GU 1023/1. The third author was supported by the DFG, projects ITKO, NI 369/5, and AREG, NI 369/9.

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 409-420
http://drops.dagstuhl.de/opus/volltexte/2009/1820

1. Introduction

Nemhauser and Trotter [20] proved a famous theorem in combinatorial optimization. In terms of the NP-hard Vertex Cover[1] problem, it can be formulated as follows:

NT-Theorem [20, 4]. *For an undirected graph $G = (V, E)$ one can compute in polynomial time two disjoint vertex subsets A and B, such that the following three properties hold:*

(1) If S' is a vertex cover of the induced subgraph $G[V \setminus (A \cup B)]$, then $A \cup S'$ is a vertex cover of G.
(2) There is a minimum-cardinality vertex cover S of G with $A \subseteq S$.
(3) Every vertex cover of the induced subgraph $G[V \setminus (A \cup B)]$ has size at least $|V \setminus (A \cup B)|/2$.

In other words, the NT-Theorem provides a polynomial-time data reduction for Vertex Cover. That is, for vertices in A it can already be decided in polynomial time to put them into the solution set and vertices in B can be ignored for finding a solution. The NT-Theorem is very useful for approximating Vertex Cover. The point is that the search for an approximate solution can be restricted to the induced subgraph $G[V \setminus (A \cup B)]$. The NT-Theorem directly delivers a factor-2 approximation for Vertex Cover by choosing $V \setminus B$ as the vertex cover. Chen et al. [7] first observed that the NT-Theorem directly yields a $2k$-vertex problem kernel for Vertex Cover, where the parameter k denotes the size of the solution set. Indeed, this is in a sense an "ultimate" kernelization result in parameterized complexity analysis [10, 11, 21] because there is good reason to believe that there is a matching lower bound $2k$ for the kernel size unless P=NP [16].

Since its publication numerous authors have referred to the importance of the NT-Theorem from the viewpoint of polynomial-time approximation algorithms (e.g., [4, 17]) as well as from the viewpoint of parameterized algorithmics (e.g., [1, 7, 9]). The relevance of the NT-Theorem comes from both its practical usefulness in solving the Vertex Cover problem as well as its theoretical depth having led to numerous further studies and follow-up work [1, 4, 9]. In this work, our main contribution is to provide a more general and more widely applicable version of the NT-Theorem. The corresponding algorithmic strategies and proof techniques, however, are not achieved by a generalization of known proofs of the NT-Theorem but are completely different and are based on extremal combinatorial arguments. Vertex Cover can be formulated as the problem of finding a minimum-cardinality set of vertices whose deletion makes a graph edge-free, that is, the remaining vertices have degree 0. Our main result is to prove a generalization of the NT-Theorem that helps in finding a minimum-cardinality set of vertices whose deletion leaves a graph of maximum degree d for arbitrary but fixed d. Clearly, $d = 0$ is the special case of Vertex Cover.

Motivation. Since the NP-hard Bounded-Degree Deletion problem—given a graph and two positive integers k and d, find at most k vertices whose deletion leaves a graph of maximum vertex degree d—stands in the center of our considerations, some more explanations about its relevance follow. Bounded-Degree Deletion (or its dual problem) already appears in some theoretical work, e.g., [6, 18, 22], but so far it has received considerably less attention than Vertex Cover, one of the best studied problems in combinatorial optimization [17]. To advocate and justify more research on Bounded-Degree Deletion,

[1] Vertex Cover is the following problem: Given an undirected graph, find a minimum-cardinality set S of vertices such that each edge has at least one endpoint in S.

we describe an application in computational biology. In the analysis of genetic networks based on micro-array data, recently a clique-centric approach has shown great success [3, 8]. Roughly speaking, finding cliques or near-cliques (called paracliques [8]) has been a central tool. Since finding cliques is computationally hard (also with respect to approximation), Chesler et al. [8, page 241] state that "cliques are identified through a transformation to the complementary dual VERTEX COVER problem and the use of highly parallel algorithms based on the notion of fixed-parameter tractability." More specifically, in these VERTEX COVER-based algorithms polynomial-time data reduction (such as the NT-Theorem) plays a decisive role [19] (also see [1]) for efficient solvability of the given real-world data. However, since biological and other real-world data typically contain errors, the demand for finding cliques (that is, fully connected subgraphs) often seems overly restrictive and somewhat relaxed notations of cliques are more appropriate. For instance, Chesler et al. [8] introduced paracliques, which are achieved by greedily extending the found cliques by vertices that are connected to almost all (para)clique vertices. An elegant mathematical concept of "relaxed cliques" is that of s-plexes[2] where one demands that each s-plex vertex does not need to be connected to all other vertices in the s-plex but to all but $s-1$. Thus, cliques are 1-plexes. The corresponding problem to find maximum-cardinality s-plexes in a graph is basically as computationally hard as clique detection is [2, 18]. However, as VERTEX COVER is the dual problem for clique detection, BOUNDED-DEGREE DELETION is the dual problem for s-plex detection: An n-vertex graph has an s-plex of size k iff its complement graph has a solution set for BOUNDED-DEGREE DELETION with $d = s-1$ of size $n-k$, and the solution sets can directly be computed from each other. The VERTEX COVER polynomial-time data reduction algorithm has played an important role in the practical success story of analyzing real-world genetic and other biological networks [3, 8]. Our new polynomial-time data reduction algorithms for BOUNDED-DEGREE DELETION have the potential to play a similar role.

Our results. Our main theorem can be formulated as follows.

BDD-DR-Theorem (Theorem 2). *For an undirected n-vertex and m-edge graph $G = (V, E)$, we can compute two disjoint vertex subsets A and B in $O(n^{5/2} \cdot m + n^3)$ time, such that the following three properties hold:*

(1) If S' is a solution set for BOUNDED-DEGREE DELETION *of the induced subgraph $G[V \setminus (A \cup B)]$, then $S := S' \cup A$ is a solution set for* BOUNDED-DEGREE DELETION *of G.*

(2) There is a minimum-cardinality solution set S for BOUNDED-DEGREE DELETION *of G with $A \subseteq S$.*

(3) Every solution set for BOUNDED-DEGREE DELETION *of the induced subgraph $G[V \setminus (A \cup B)]$ has size at least*

$$\frac{|V \setminus (A \cup B)|}{d^3 + 4d^2 + 6d + 4}.$$

In terms of parameterized algorithmics, this gives a $(d^3 + 4d^2 + 6d + 4) \cdot k$-vertex problem kernel for BOUNDED-DEGREE DELETION, which is linear in k for constant d-values, thus joining a number of other recent "linear kernelization results" [5, 12, 14, 15]. Our general result specializes to a $4k$-vertex problem kernel for VERTEX COVER (the NT-Theorem provides a size-$2k$ problem kernel), but applies to a larger class of problems.

[2]Introduced in 1978 by Seidman and Foster [24] in the context of social network analysis. Recently, this concept has again found increased interest [2, 18].

For instance, a slightly modified version of the BDD-DR-Theorem (with essentially the same proof) yields a $15k$-vertex problem kernel for the problem of packing at least k vertex-disjoint length-2 paths of an input graph, giving the same bound as shown in work focussing on this problem [23].[3] For the problem, where, given an undirected graph, one seeks a set of at least k vertex-disjoint stars[4] of the same constant size, we show that a kernel with a linear number of vertices can be achieved, improving the best previous quadratic kernelization [23]. We emphasize that our data reduction technique is based on extremal combinatorial arguments; the resulting combinatorial kernelization algorithm has practical potential and implementation work is underway. Note that for $d = 0$ our algorithm computes the same type of structure as in the "crown decomposition" kernelization for Vertex Cover (see, for example, [1]). However, for $d \geq 1$ the structure returned by our algorithm is much more complicated; in particular, unlike for Vertex Cover crown decompositions, in the BDD-DR-Theorem the set A is not necessarily a separator and the set B does not necessarily form an independent set.

Exploring the borders of parameterized tractability of Bounded-Degree Deletion for arbitrary values of the degree value d, we show the following.

Theorem 1. For unbounded d (given as part of the input), Bounded-Degree Deletion is $W[2]$-complete with respect to the parameter k denoting the number of vertices to delete.

In other words, there is no hope for fixed-parameter tractability with respect to the parameter k in the case of unbounded d-values. Due to the lack of space the proof of Theorem 1 and several proofs of lemmas needed to show Theorem 2 are omitted.

2. Preliminaries

A *bdd-d-set* for a graph $G = (V, E)$ is a vertex subset whose removal from G yields a graph in which each vertex has degree at most d. The central problem of this paper is

Bounded-Degree Deletion
Input: An undirected graph $G = (V, E)$, and integers $d \geq 0$ and $k > 0$.
Question: Does there exist a bdd-d-set $S \subseteq V$ of size at most k for G?

In this paper, for a graph $G = (V, E)$ and a vertex set $S \subseteq V$, let $G[S]$ be the subgraph of G induced by S and $G - S := G[V \setminus S]$. The *open neighborhood* of a vertex v or a vertex set $S \subseteq V$ in a graph $G = (V, E)$ is denoted as $N_G(v) := \{u \in V \mid \{u, v\} \in E\}$ and $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$, respectively. The *closed neighborhood* is denoted as $N_G[v] := N_G(v) \cup \{v\}$ and $N_G[S] := N_G(S) \cup S$. We write $V(G)$ and $E(G)$ to denote the vertex and edge set of G, respectively. A *packing* P of a graph G is a set of pairwise vertex-disjoint subgraphs of G. A graph has maximum degree d when every vertex in the graph has degree at most d. A graph property is called *hereditary* if every induced subgraph of a graph with this property has the property as well.

Parameterized algorithmics [10, 11, 21] is an approach to finding optimal solutions for NP-hard problems. A common method in parameterized algorithmics is to provide polynomial-time executable *data reduction rules* that lead to a *problem kernel* [13]. This is the most important concept for this paper. Given a parameterized problem instance (I, k), a

[3] Very recently, Wang et al. [25] improved the $15k$-bound to a $7k$-bound. We claim that our kernelization based on the BDD-DR-Theorem method can be easily adapted to also deliver the $7k$-bound.

[4] A *star* is a tree where all of the vertices but one are leaves.

data reduction rule replaces (I, k) by an instance (I', k') in polynomial time such that $|I'| \leq |I|$, $k' \leq k$, and (I, k) is a Yes-instance if and only if (I', k') is a Yes-instance. A parameterized problem is said to have a problem kernel, or, equivalently, *kernelization*, if, after the exhaustive application of the data reduction rules, the resulting reduced instance has size $f(k)$ for a function f depending only on k. Roughly speaking, the kernel size $f(k)$ plays a similar role in the subject of problem kernelization as the approximation factor plays for approximation algorithms.

3. A Local Optimization Algorithm for Bounded-Degree Deletion

The main result of this section is the following generalization of the Nemhauser-Trotter-Theorem [20] for Bounded-Degree Deletion with constant d.

Theorem 2 (BDD-DR-Theorem)**.** For an n-vertex and m-edge graph $G = (V, E)$, we can compute two disjoint vertex subsets A and B in $O(n^{5/2} \cdot m + n^3)$ time, such that the following three properties hold:

(1) If S' is a bdd-d-set of $G - (A \cup B)$, then $S := S' \cup A$ is a bdd-d-set of G.
(2) There is a minimum-cardinality bdd-d-set S of G with $A \subseteq S$.
(3) Every bdd-d-set of $G - (A \cup B)$ has size at least $\frac{|V \setminus (A \cup B)|}{d^3+4d^2+6d+4}$.

This first two properties are called the *local optimality conditions*. The remainder of this section is dedicated to the proof of this theorem. More specifically, we present an algorithm called **compute_AB** (see Figure 1) which outputs two sets A and B fulfilling the three properties given in Theorem 2. The core of this algorithm is the procedure **find_extremal** (see Figure 2) running in $O(n^{3/2} \cdot m + n^2)$ time. This procedure returns two disjoint vertex subsets C and D that, among others, satisfy the local optimality conditions. The procedure is iteratively called by **compute_AB**. The overall output sets A and B then are the union of the outputs of all applications of **find_extremal**. Actually, **find_extremal** searches for $C \subseteq V$, $D \subseteq V$, $C \cap D = \emptyset$ satisfying the following two conditions:

C1 Each vertex in $N_G[D] \setminus C$ has degree at most d in $G - C$, and
C2 C is a minimum-cardinality bdd-d-set for $G[C \cup D]$.

It is not hard to see that these two conditions are stronger than the local optimality conditions of Theorem 2:

Lemma 1. Let C and D be two vertex subsets satisfying conditions C1 and C2. Then, the following is true:

(1) If S' is a bdd-d-set of $G - (C \cup D)$, then $S := S' \cup C$ is a bdd-d-set of G.
(2) There is a minimum-cardinality bdd-d-set S of G with $C \subseteq S$.

Lemma 1 will be used in the proof of Theorem 2—it helps to make the description of the underlying algorithm and the corresponding correctness proofs more accessible. As a direct application of Theorem 2, we get the following corollary.

Corollary 1. Bounded-Degree Deletion with constant d admits a problem kernel with at most $(d^3 + 4d^2 + 6d + 4) \cdot k$ vertices, which is computable in $O(n^{5/2} \cdot m + n^3)$ time.

We use the following easy-to-verify forbidden subgraph characterization of bounded-degree graphs: A graph G has maximum degree d if and only if there is no "$(d + 1)$-star" in G.

Algorithm: compute_AB (G)
Input: An undirected graph G.
Output: Vertex subsets A and B satisfying the three properties of Theorem 2.

1 $A := \emptyset, B := \emptyset$
2 Compute a witness X and the corresponding residual $Y := V \setminus X$ for G
3 **If** $|Y| \leq (d+1)^2 \cdot |X|$ **then return** (A, B)
4 $(C, D) \leftarrow$ **find_extremal** (G, X, Y).
5 $G \leftarrow G - (C \cup D); A \leftarrow A \cup C; B \leftarrow B \cup D;$ **goto** line 2

Figure 1: Pseudo-code of the main algorithm for computing A and B.

Definition 3.1. For $s \geq 1$, the graph $K_{1,s} = (\{u, v_1, \ldots, v_s\}, \{\{u, v_1\}, \ldots, \{u, v_s\}\})$ is called an *s-star*. The vertex u is called the *center* of the star. The vertices $v_1, \ldots, v_s$ are the *leaves* of the star. A $\leq s$*-star* is an s'-star with $s' \leq s$.

Due to this forbidden subgraph characterization of bounded-degree graphs, we can also derive a linear kernelization for the $(d+1)$-Star Packing problem. In this problem, given an undirected graph, one seeks for at least k vertex-disjoint $(d+1)$-stars for a constant d. With a slight modification of the proof of Theorem 2, we get the following corollary.

Corollary 2. $(d+1)$-Star Packing admits a problem kernel with at most $(d^3 + 4d^2 + 6d + 4) \cdot k$ vertices, which is computable in $O(n^{5/2} \cdot m + n^3)$ time.

For $d \geq 2$, the best known kernelization result was a $O(k^2)$ kernel [23]. Note that the special case of $(d+1)$-Star Packing with $d = 1$ is also called P_3-Packing, a problem well-studied in the literature, see [23, 25]. Corollary 2 gives a $15k$-vertex problem kernel. The best-known bound is $7k$ [25]. However, the improvement from the formerly best bound $15k$ [23] is achieved by improving a properly defined witness structure by local modifications. This trick also works with our approach, that is, we can show that the NT-like approach also yields a $7k$-vertex problem kernel for 2-Star Packing.

3.1. The Algorithm

We start with an informal description of the algorithm. As stated in the introduction of this section, the central part is Algorithm **compute_AB** shown in Figure 1.

Using the characterization of bounded-degree graphs by forbidding large stars, in line 2 **compute_AB** starts with computing two vertex sets X and Y: First, with a straightforward greedy algorithm, compute a *maximal* $(d+1)$*-star packing* of G, that is, a set of vertex-disjoint $(d+1)$-stars that cannot be extended by adding another $(d+1)$-star. Let X be the set of vertices of the star packing. Since the number of stars in the packing is a lower bound for the size of a minimum bdd-d-set, X is a factor-$(d+2)$ approximate bdd-d-set. Greedily remove vertices from X such that X is still a bdd-d-set, and finally set $Y := V \setminus X$. We call X the *witness* and Y the corresponding *residual*.

If the residual Y is too big (condition in line 3), the sets X and Y are passed in line 4 to the procedure **find_extremal** in Figure 2 which computes two sets C and D satisfying conditions C1 and C2. Computing X and Y represents the first step to find a subset pair satisfying condition C1: Since there is no vertex that has degree more than d in $G - X$ (due

Procedure: find_extremal (G, X, Y)
Input: An undirected graph G, witness X, and residual Y.
Output: Vertex subsets C and D satisfying the local optimality conditions.

1 $J \leftarrow$ bipartite graph with X and Y as its two vertex subsets and $E(J) \leftarrow \{\{u, v\} \in E(G) \mid u \in X \text{ and } v \in Y\}$
2 $F_0^X \leftarrow \emptyset$ ▷ *Initialize empty set of forbidden vertices*
3 start with $j = 0$ and **while** $F_j^X \neq X$ **do** ▷ *Loop while not all vertices in X are forbidden*
4 $F_j^Y \leftarrow N_G[N_J(F_j^X)] \setminus X$ ▷ *Determine forbidden vertices in Y*
5 $P \leftarrow$ **star-packing**$(J - (F_j^X \cup F_j^Y), X \setminus F_j^X, Y \setminus F_j^Y, d)$
6 $D_0 \leftarrow Y \setminus (F_j^Y \cup V(P))$ ▷ *Vertices in Y that are not forbidden and not in P*
7 start with $i = 0$ and **repeat** ▷ *Start search for C, D satisfying C2*
8 $C_i \leftarrow N_J(D_i)$
9 $D_{i+1} \leftarrow N_P(C_i) \cup D_i$
10 $i \leftarrow i + 1$
11 **until** $D_i = D_{i-1}$
12 $C \leftarrow C_i$, $D \leftarrow D_i$
13 **if** $C = X \setminus F_j^X$ **then** ▷ *C, D also satisfy C1*
14 **return** (C, D)
15 $F_{j+1}^X \leftarrow X \setminus C$ ▷ *Determine forbidden vertices in X for next iteration*
16 $j \leftarrow j + 1$
17 **end while**
18 $F_j^Y \leftarrow N_G[N_J(F_j^X)] \setminus X$ ▷ *Recompute forbidden vertices in Y (as in line 4)*
19 **return** $(\emptyset, V \setminus (X \cup F_j^Y))$

Procedure: star-packing (J, V_1, V_2, d)
Input: A bipartite graph J with two vertex subsets V_1 and V_2.
Output: A maximum-edge packing of stars that have their centers in V_1 and have at most $d + 1$ leaves in V_2.
See Lemma 2, the straightforward implementation details using matching techniques are omitted.

Figure 2: Pseudo-code of the procedure computing the intermediary vertex subset pair (C, D).

to the fact that X is a bdd-d-set), the search is limited to those subset pairs where C is a subset of the witness X and D is a subset of Y.

Algorithm **compute_AB** calls **find_extremal** iteratively until the sets A and B, which are constructed by the union of the outputs of all applications of **find_extremal** (see line 5), satisfy the third property in Theorem 2. In the following, we intuitively describe the basic ideas behind **find_extremal**.

To construct the set C from X, we compute again a star packing P with the centers of the stars being from X and the leaves being from Y. We relax, on the one hand, the requirement that the stars in the packing have exactly $d + 1$ leaves, that is, the packing P might contain $\leq d$-stars. On the other hand, P should have a maximum number of edges. The rough idea behind the requirement for a maximum number of edges is to maximize the

number of $(d+1)$-stars in P in the course of the algorithm. Moreover, we can observe that, by setting C equal to the center set of the $(d+1)$-stars in P and D equal to the leaf set of the $(d+1)$-stars in P, C is a minimum bdd-d-set of $G[C \cup D]$ (condition C2). We call such a packing a *maximum-edge X-center $\leq (d+1)$-star packing*. For computing P, the algorithm constructs an auxiliary bipartite graph J with X as one vertex subset and Y as the other. The edge set of J consists of the edges in G with exactly one endpoint in X. See line 1 of Figure 2. Obviously, a maximum-edge X-center $\leq (d+1)$-star packing of G corresponds one-to-one with a maximum-edge packing of stars in J that have their centers in X and have at most $d+1$ leaves in the other vertex subset. Then, the star packing P can be computed by using techniques for computing maximum matchings in J (in the following, let **star-packing**(J,V_1,V_2,d) denote an algorithm that computes a maximum-edge V_1-center $\leq (d+1)$-star packing P on the bipartite graph J).

The most involved part of **find_extremal** in Figure 2 is to guarantee that the output subsets in line 4 fulfill condition C1. To this end, one uses an iterative approach to compute the star packing P. Roughly speaking, in each iteration, if the subsets C and D do not fulfill condition C1, then exclude from further iterations the vertices from D that themselves or whose neighbors violate this condition. See lines 2 to 15 of Figure 2 for more details of the iterative computation. Herein, for $j \geq 0$, the sets $F_j^X \subseteq X$ and $F_j^Y \subseteq Y$, where F_j^X is initialized with the empty set, and F_j^Y is computed using F_j^X, store the vertices excluded from computing P. To find the vertices that themselves cause the violation of the condition, that is, vertices in D that have neighbors in $X \setminus C$, one uses an augmenting path computation in lines 7 to 11 to get in line 12 subsets C and D such that the vertices in D do *not* themselves violate the condition. Roughly speaking, the existence of an edge e from some vertex in D to some vertex in $X \setminus C$ would imply that the $\leq (d+1)$-star packing is not maximum (witnessed by an augmenting path beginning with e—in principle, this idea is also used for finding crown decompositions, cf. [1]). The vertices whose neighbors cause the violation of condition C1 are all vertices in D with neighbors in $Y \setminus D$ that themselves have neighbors in $X \setminus C$. These neighbors in $Y \setminus D$ and the corresponding vertices in D are excluded in line 4 and line 18. We will see that the number of all excluded vertices is $O(|X \setminus C|)$, thus, in total, we do not exclude too many vertices with this iterative method. The formal proof of correctness is given in the following subsection.

3.2. Running Time and Correctness

Now, we show that **compute_AB** in Figure 1 computes in the claimed time two vertex subsets A and B that fulfill the three properties given in Theorem 2.

3.2.1. *Running Time of **find_extremal**.* We begin with the proof of the running time of the procedure **find_extremal** in Figure 2, which uses the following lemmas.

Lemma 2. Procedure **star-packing**(J, V_1, V_2, d) in Figure 2 runs in $O(\sqrt{n} \cdot m)$ time.

The next lemma is also used for the correctness proof; in particular, it guarantees the termination of the algorithm.

Lemma 3. If the condition in line 13 of Figure 2 is false for a $j \geq 0$, then $F_j^X \subsetneq F_{j+1}^X$.

Proof. In lines 4 and 5 of Figure 2, all vertices in F_j^X and their neighbors $N_J(F_j^X)$ are excluded from the star packing P in the jth iteration of the outer loop. Moreover, the vertices in $N_J(F_j^X)$ are excluded from the set D_0 (line 6). Therefore, a vertex in F_j^X cannot be added to C in line 12. Thus F_{j+1}^X (set to $X \setminus C$ in line 15) contains F_j^X. Moreover, this containment is proper, as otherwise the condition in line 13 would be true. ∎

Lemma 4. Procedure **find_extremal** runs in $O(n^{3/2} \cdot m + n^2)$ time.

3.2.2. *Correctness of **find_extremal**.* The correctness proof for **find_extremal** in Figure 2 is more involved than its running time analysis. The following lemmas provide some properties of (C, D) which are needed.

Lemma 5. For each $j \geq 0$ the following properties hold after the execution of line 12 in Figure 2:

(1) every vertex in C is a center vertex of a $(d+1)$-star in P, and
(2) the leaves of every star in P with center in C are vertices in D.

Proof. (Sketch) To prove (1), first of all, we show that $v \in C$ implies $v \in V(P)$, since, otherwise, we could get a P-augmenting path from some element in D_0 to v. A P-augmenting path is a path where the edges in $E(P)$ and the edges not in $E(P)$ alternate, and the first and the last edge are not in $E(P)$. This P-augmenting path can be constructed in an inductive way by simulating the construction of C_i in lines 6 to 11 of Figure 2. From this P-augmenting path, we can then construct a X-center $\leq (d+1)$-star packing that has more edges than P, contradicting that $E(P)$ has maximum cardinality. Second, every vertex in C is a center of a star due to the definition of P and Procedure **star-packing**. Finally, if a vertex $v \in C$ is the center of a star with less than $(d+1)$ leaves, then again we get a P-augmenting path from some element in D_0 to v.

The second statement follows easily from Procedure **star-packing** and the pseudo-code in lines 6 to 12. ∎

Lemma 6. For each $j \geq 0$ there is no edge in G between D and $N_J(F_j^X)$.

Proof. The vertices in F_j^X and the vertices in $N_G[N_J(F_j^X)] \setminus X$ are excluded from the computation of P and are not contained in D_0 (lines 4 to 6 in Figure 2). Thus, $N_J[F_j^X] \cap D = \emptyset$ and therefore there are no edges in G between D and $N_J(F_j^X)$. ∎

The next lemma shows that the output of **find_extremal** fulfills the local optimality conditions.

Lemma 7. Procedure **find_extremal** returns two disjoint vertex subsets fulfilling conditions C1 and C2.

Proof. Clearly, the output consists of two disjoints sets. The algorithm returns in lines 14 or 19 of Figure 2. If it returns in line 19, then the output C is empty and D contains only vertices that have a distance at least 3 to the vertices in X: The condition in line 3 implies $F_j^X = X$ and, therefore, F_j^Y contains all vertices in $G \setminus X$ that have distance at most 2 to the vertices in X. Since X is a bdd-d-set of G, all vertices in D and their neighbors in G have a degree at most d. This implies that both conditions hold for the output returned in this line. It remains to consider the output returned in line 14.

To show that condition C1 holds, recall that $G-X$ has maximum degree d and that $C \subseteq X$. Therefore, if for a vertex v in $V \setminus X$ we have $N_J(v) \subseteq C$, then v has degree at most d in $G - C$. Thus, to show that each vertex in $N_G[D] \setminus C$ has degree at most d in $G - C$, it suffices to prove that $N_J(N_G[D] \setminus C) \subseteq C$. We show separately that $N_J(D) \subseteq C$ and that $N_J(N_G(D) \setminus C) \subseteq C$.

The assignment in line 8 and the until-condition in line 11 directly give $N_J(D) \subseteq C$. Due to Lemma 6 there is no edge in G between D and $N_J(F_j^X)$, where $F_j^X = X \setminus C$ (the if-condition in line 13, which has to be satisfied for the procedure to return in line 14). From this it follows that the vertices in $N_G(D) \setminus C$ have no vertex in F_j^X as neighbor and, thus, $N_J(N_G(D) \setminus C) \cap F_j^X = \emptyset$. Therefore, $N_J(N_G(D) \setminus C) \subseteq C$.

By Properties 1 and 2 of Lemma 5, there are exactly $|C|$ many vertex-disjoint $(d+1)$-stars in $G[C \cup D]$. Moreover, there is no $(d+1)$-star in $G[D]$, since X is a bdd-d-set of G. Thus, C is a minimum-cardinality bdd-d-set of $G[C \cup D]$. ∎

3.2.3. *Running Time and Correctness of* ***compute_AB***. To prove the running time and correctness of **compute_AB**, we have to show that the output of **find_extremal** contains sufficiently many vertices of Y. To this end, the following lemma plays a decisive role.

Lemma 8. For all $j \geq 0$, the set F_j^Y in line 4 and line 18 of Figure 2 has size at most $(d+1)^2 \cdot |F_j^X|$.

Proof. The proof is by induction on j. The claim trivially holds for $j = 0$, since $F_0^Y = \emptyset$. Assume that the claim is true for $j > 0$. Since $F_j^X \subsetneq F_{j+1}^X$ (Lemma 3), we have

$$F_{j+1}^Y = F_j^Y \cup N_{G-X}[N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X)].$$

We first bound the size of $N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X)$. Since F_{j+1}^X was set to $X \setminus C$ at the end of the jth iteration of the outer loop (line 15), the vertices in $N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X)$ were not excluded from computing the packing P (line 5) of the jth iteration. Moreover, $N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X) \subseteq V(P)$ for the star packing P computed in the jth iteration, since, otherwise, the set D_0 in line 6 would contain a vertex v in $N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X)$ and, then, line 8 would include $N_J(v)$ into C, which would contradict the fact that $C \cap F_{j+1}^X = \emptyset$ (line 15). Due to property 2 in Lemma 5 the leaves of every star in P with center in C are vertices in D and, thus, the vertices in $N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X)$ are leaves of stars in P with centers in $F_{j+1}^X \setminus F_j^X$. Since each star has at most $(d+1)$ leaves, the set $N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X)$ has size at most $(d+1) \cdot |F_{j+1}^X \setminus F_j^X|$. The remaining part is easy to bound: since all the vertices in $V \setminus X$ have degree at most d, we get

$$\begin{aligned}\left|N_{G-X}[N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X)]\right| &\leq (d \cdot (d+1) + (d+1)) \cdot |F_{j+1}^X \setminus F_j^X| \\ &= (d+1)^2 \cdot |F_{j+1}^X \setminus F_j^X|.\end{aligned}$$

With the induction hypothesis, we get that

$$\begin{aligned}|F_{j+1}^Y| &\leq |F_j^Y| + |N_{G-X}[N_{J-F_j^Y}(F_{j+1}^X \setminus F_j^X)]| \\ &= (d+1)^2 \cdot |F_j^X| + (d+1)^2 \cdot |F_{j+1}^X \setminus F_j^X| = (d+1)^2 \cdot |F_{j+1}^X|.\end{aligned}$$

∎

Lemma 9. Procedure **find_extremal** always finds two sets C and D such that $|Y \setminus D| \leq (d+1)^2 \cdot |X \setminus C|$.

Proof. If **find_extremal** terminates, then $V' = F_j^X \cup F_j^Y$ for the graph $G' = (V', E')$ resulting by removing $C \cup D$ from G. Since $C \subseteq X$ and $D \subseteq Y$, we have $X \setminus C = F_j^X$ and $Y \setminus D = F_j^Y$, and by Lemma 8 it follows immediately that $|Y \setminus D| \leq (d+1)^2 \cdot |X \setminus C|$. ∎

Therefore, if $|Y| > (d+1)^2 \cdot |X|$, then **find_extremal** always returns two sets C and D such that D is not empty.

Lemma 10. Algorithm **compute_AB** runs in $O(n^{5/2} \cdot m + n^3)$ time.

Lemma 11. The sets A and B computed by **compute_AB** fulfill the three properties given in Theorem 2.

Proof. Since every (C, D) output by **find_extremal** in line 4 of **compute_AB** in Figure 1 fulfills conditions C1 and C2 (Lemma 7), the pair (A, B) output in line 3 of **compute_AB** fulfills conditions C1 and C2, and, therefore, also the local optimality conditions (Lemma 1). It remains to show that (A, B) fulfills the size condition.

Let X and Y be the last computed witness and residual, respectively. Since the condition in line 3 is true, we know that $|Y| \leq (d+1)^2 \cdot |X|$. Recall that X is a factor-$(d+2)$ approximate bdd-d-set for $G' := G - (A \cup B)$. Thus, every bdd-d-set of G' has size at least $|X|/(d+2)$. Since the output sets A and B fulfill the local optimality conditions and the bounded-degree property is hereditary, every bdd-d-set of G' has size at least

$$\frac{|X|}{d+2} \overset{(*)}{\geq} \frac{|V'|}{(d+2)((d+1)^2+1)} = \frac{|V'|}{(d^3+4d^2+6d+4)}.$$

The inequality (*) follows from the fact that Y is small, that is, $|Y| \leq (d+1)^2 \cdot |X|$ (note that $V' = X \cup Y$). ∎

With Lemmas 10 and 11, the proof of Theorem 2 is completed.

4. Conclusion

Our main result is to generalize the Nemhauser-Trotter-Theorem, which applies to the Bounded-Degree Deletion problem with $d = 0$ (that is, Vertex Cover), to the general case with arbitrary $d \geq 0$. In particular, in this way we contribute problem kernels with a number of vertices linear in the solution size k for all constant values of d for Bounded-Degree Deletion. To this end, we developed a new algorithmic strategy that is based on extremal combinatorial arguments. The original NT-Theorem [20] has been proven using linear programming relaxations—we see no way how this could have been generalized to Bounded-Degree Deletion. By way of contrast, we presented a purely combinatorial data reduction algorithm which is also completely different from known combinatorial data reduction algorithms for Vertex Cover (see [1, 4, 9]). Finally, Baldwin et al. [3, page 175] remarked that, with respect to practical applicability in the case of Vertex Cover kernelization, combinatorial data reduction algorithms are more powerful than "slower methods that rely on linear programming relaxation". Hence, we expect that benefits similar to those derived from Vertex Cover kernelization for biological network analysis (see the motivation part of our introductory discussion) may be provided by Bounded-Degree Deletion kernelization.

References

[1] F. N. Abu-Khzam, M. R. Fellows, M. A. Langston, and W. H. Suters. Crown structures for vertex cover kernelization. *Theory Comput. Syst.*, 41(3):411–430, 2007.

[2] B. Balasundaram, S. Butenko, I. V. Hicks, and S. Sachdeva. Clique relaxations in social network analysis: The maximum k-plex problem. Manuscript, 2008.

[3] N. Baldwin, E. Chesler, S. Kirov, M. Langston, J. Snoddy, R. Williams, and B. Zhang. Computational, integrative, and comparative methods for the elucidation of genetic coexpression networks. *Journal of Biomedicine and Biotechnology*, 2(2005):172–180, 2005.

[4] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Ann. of Discrete Math.*, 25:27–45, 1985.

[5] H. L. Bodlaender and E. Penninkx. A linear kernel for planar feedback vertex set. In *Proc. 3rd IWPEC*, volume 5018 of *LNCS*, pages 160–171. Springer, 2008.

[6] H. L. Bodlaender and B. van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Inform. and Comput.*, 167(2):86–119, 2001.

[7] J. Chen, I. A. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[8] E. J. Chesler, L. Lu, S. Shou, Y. Qu, J. Gu, J. Wang, H. C. Hsu, J. D. Mountz, N. E. Baldwin, M. A. Langston, D. W. Threadgill, K. F. Manly, and R. W. Williams. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nature Genetics*, 37(3):233–242, 2005.

[9] M. Chlebík and J. Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Appl. Math.*, 156:292–312, 2008.

[10] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[11] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[12] J. Guo. A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.*, 2008. To appear.

[13] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[14] J. Guo and R. Niedermeier. Linear problem kernels for NP-hard problems on planar graphs. In *Proc. 34th ICALP*, volume 4596 of *LNCS*, pages 375–386. Springer, 2007.

[15] I. A. Kanj, M. J. Pelsmajer, G. Xia, and M. Schaefer. On the induced matching problem. *J. Comput. System Sci.*, 2009. To appear.

[16] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *J. Comput. System Sci.*, 74(3):335–349, 2008.

[17] S. Khuller. The Vertex Cover problem. *ACM SIGACT News*, 33(2):31–33, 2002.

[18] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for enumerating dense subgraphs. In *Proc. 13th COCOON*, volume 4598 of *LNCS*, pages 140–150. Springer, 2007.

[19] M. A. Langston, 2008. Personal communication.

[20] G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Math. Program.*, 8:232–248, 1975.

[21] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[22] N. Nishimura, P. Ragde, and D. M. Thilikos. Fast fixed-parameter tractable algorithms for nontrivial generalizations of Vertex Cover. *Discrete Appl. Math.*, 152(1–3):229–245, 2005.

[23] E. Prieto and C. Sloper. Looking at the stars. *Theor. Comput. Sci.*, 351(3):437–445, 2006.

[24] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology*, 6:139–154, 1978.

[25] J. Wang, D. Ning, Q. Feng, and J. Chen. An improved parameterized algorithm for a generalized matching problem. In *Proc. 5th TAMC*, volume 4978 of *LNCS*, pages 212–222. Springer, 2008.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 421–432
www.stacs-conf.org

KERNEL(S) FOR PROBLEMS WITH NO KERNEL: ON OUT-TREES WITH MANY LEAVES (EXTENDED ABSTRACT)

HENNING FERNAU [1] AND FEDOR V. FOMIN [2] AND DANIEL LOKSHTANOV [2] AND DANIEL RAIBLE [1] AND SAKET SAURABH [2] AND YNGVE VILLANGER [2]

[1] Univ. Trier, FB 4, Abteilung Informatik, 54286 Trier, Germany.
E-mail address: {fernau|raible}@uni-trier.de

[2] Department of Informatics, University of Bergen, Bergen Norway.
E-mail address: {fedor.fomin|daniello|saket.saurabh|yngve.villanger}@ii.uib.no

Abstract. The k-Leaf Out-Branching problem is to find an out-branching, that is a rooted oriented spanning tree, with at least k leaves in a given digraph. The problem has recently received much attention from the viewpoint of parameterized algorithms. Here, we take a kernelization based approach to the k-Leaf-Out-Branching problem. We give the first polynomial kernel for Rooted k-Leaf-Out-Branching, a variant of k-Leaf-Out-Branching where the root of the tree searched for is also a part of the input. Our kernel has cubic size and is obtained using extremal combinatorics.

For the k-Leaf-Out-Branching problem, we show that no polynomial kernel is possible unless the polynomial hierarchy collapses to third level by applying a recent breakthrough result by Bodlaender et al. (ICALP 2008) in a non-trivial fashion. However, our positive results for Rooted k-Leaf-Out-Branching immediately imply that the seemingly intractable k-Leaf-Out-Branching problem admits a data reduction to n independent $O(k^3)$ kernels. These two results, tractability and intractability side by side, are the first ones separating *many-to-one kernelization* from *Turing kernelization*. This answers affirmatively an open problem regarding "cheat kernelization" raised by Mike Fellows and Jiong Guo independently.

1. Introduction

Parameterized decision problems are defined by specifying the input (I), the parameter (k), and the question to be answered. A parameterized problem that can be solved in time $f(k)|I|^{O(1)}$ where f is a function of k alone is said to be fixed parameter tractable (FPT). Kernelization is a powerful and natural technique in the design of parameterized algorithms. The main idea of kernelization is to replace a given parameterized instance (I, k) of a problem Π by a simpler instance (I', k') of Π in polynomial time, such that (I, k)

Key words and phrases: Parameterized Algorithms, Kernelization, Out-Branching, Max-Leaf, Lower Bounds.

The authors gratefully acknowledge the support given by a German-Norwegian research grant.

is a yes-instance if and only if (I', k') is a yes-instance and the size of I' is bounded by a function of k alone. The reduced instance I' is called the *kernel* for the problem. Typically kernelization algorithms work by applying reduction rules, which iteratively reduce the instance to an equivalent "smaller" instance. From this point of view, kernelization can be seen as pre-processing with an explicit performance guarantee, "a humble strategy for coping with hard problems, almost universally employed" [14].

A parameterized problem is said to have a polynomial kernel if we have a polynomial time kernelization algorithm which reduces the size of the input instance down to a polynomial in the parameter. There are many parameterized problems for which polynomial, and even linear kernels are known [9, 8, 13, 17, 25]. Notable examples include a $2k$-sized kernel for k-Vertex Cover [9], a $O(k^2)$ kernel for k-Feedback Vertex Set [25] and a $67k$ kernel for k-Planar-Dominating Set [8], among many others. While positive kernelization results have been around for quite a while, the first results ruling out polynomial kernels for parameterized problems have appeared only recently. In a seminal paper Bodlaender et al. [4] have shown that a variety of important FPT problems cannot have polynomial kernels unless the polynomial hierarchy collapses to the third level ($PH = \Sigma_p^3$), a well known complexity theory hypothesis. Examples of such problems are k-Path, k-Minor Order Test, k-Planar Graph Subgraph Test, and many others. However, while this negative result rules out the existence of a polynomial kernel for these problems, it does not rule out the possibility of a kernelization algorithm reducing the instance to $|I|^{O(1)}$ independent polynomial kernels. This raises the question of the relationship between *many-to-one kernelization* and *Turing kernelization*, see [3, 13, 17]: Is there a natural parameterized problem having no polynomial kernel, but where we can "cheat" this lower bound by providing $|I|^{O(1)}$ polynomial kernels? Besides being of theoretical interest, this type of results would be very desirable from a practical point of view, as well. We show k-Leaf Out-Branching as the first example of such a problem.

The Maximum Leaf Spanning Tree problem on connected undirected graphs is: find a spanning tree with the maximum number of leaves in a given input graph G. The problem is well studied both from an algorithmic [16, 22, 23] and combinatorial [11, 19, 21] point of view, as well as from the parameterized complexity perspective [5, 13, 15]. An extension of Maximum Leaf Spanning Tree to directed graphs is defined as follows. We say that a subdigraph T of a digraph D is an *out-tree* if T is an oriented tree with only one vertex r of in-degree zero (called the *root*). The vertices of T of out-degree zero are called *leaves*. If T is a spanning out-tree, i.e., $V(T) = V(D)$, then T is called an *out-branching* of D. The Directed Maximum Leaf Out-Branching problem is to find an out-branching in a given digraph with the maximum number of leaves. The parameterized version of the Directed Maximum Leaf Out-Branching problem is k-Leaf Out-Branching, where for a given digraph D and integer k, it is asked to decide whether D has an out-branching with at least k leaves. If we replace "out-branching" with "out-tree" in the definition of k-Leaf Out-Branching, we get a problem called k-Leaf Out-Tree.

Unlike its undirected counterpart, the study of k-Leaf Out-Branching has only begun recently. Alon et al. [1, 2] proved that the problem is fixed parameter tractable (FPT) by providing an algorithm deciding in time $O(f(k)n)$ whether a strongly connected digraph has an out-branching with at least k leaves. Bonsma and Dorn [6] extended this result to connected digraphs, and improved the running time of the algorithm. Recently, Kneis et al. [20] provided a parameterized algorithm solving the problem in time $4^k n^{O(1)}$. This result was further improved by Daligaut et al. [10]. In a related work, Drescher

	k-Out-Tree	k-Out-Branching
Rooted	$O(k^3)$ kernel	$O(k^3)$ kernel
Unrooted	No $poly(k)$ kernel, n kernels of size $O(k^3)$	No $poly(k)$ kernel, n kernels of size $O(k^3)$

Table 1: Our Results

and Vetta [12] described an $\sqrt{OPT}$-approximation algorithm for the Directed Maximum Leaf Out-Branching problem. Let us remark that, despite similarities between directed and undirected variants of Maximum Leaf Spanning Tree, the directed case requires a totally different approach (except from [20]). However, the existence of a polynomial kernel for k-Leaf Out-Branching has not been addressed until now.

Our contribution. We prove that Rooted k-Leaf Out-Branching, where for a given vertex r one asks for a k-leaf out-branching rooted at r, admits a $O(k^3)$ kernel. A similar result also holds for Rooted k-Leaf Out-Tree, where we are looking for a rooted (not necessary spanning) tree with k leaves. While many polynomial kernels are known for undirected graphs, this is the first known non-trivial parameterized problem on digraphs admitting a polynomial kernel. To obtain the kernel we establish a number of results on the structure of digraphs not having a k-leaf out-branching. These results may be of independent interest.

In the light of our positive results it is natural to suggest that k-Leaf Out-Branching admits a polynomial kernel, as well. We find it a bit striking that this is not the case – k-Leaf Out-Branching and k-Leaf Out-Tree do not admit polynomial kernels unless $PH = \Sigma_p^3$. While the main idea of our proof is based on the framework of Bodlaender et al. [4], our adaptation is non-trivial. In particular, we use the cubic kernel obtained for Rooted k-Leaf Out-Branching to prove the lower bound. Our contributions are summarized in Table 1.

Finally, notice that the polynomial kernels for the rooted versions of our problems yield a "cheat" solution for the poly-kernel-intractable k-Leaf Out-Branching and k-Leaf Out-Tree. Let D be a digraph on n vertices. By running the kernelization for the rooted version of the problem for every vertex of D as a root, we obtain n graphs where each of them has $O(k^3)$ vertices, at least one of them having a k-leaf out-branching iff D does.

Most proofs had to be omitted due to space limitations. More information can be found at `http://arxiv.org/abs/0810.4796`.

2. Preliminaries

Let D be a directed graph or digraph for short. By $V(D)$ and $A(D)$, we represent the vertex set and arc set, respectively, of D. Given a subset $V' \subseteq V(D)$ of a digraph D, by $D[V']$ we mean the digraph induced on V'. A vertex y of D is an *in-neighbor* (*out-neighbor*) of a vertex x if $yx \in A$ ($xy \in A$). The *in-degree* (*out-degree*) of a vertex x is the number of its in-neighbors (out-neighbors) in D. Let $P = p_1p_2 \dots p_l$ be a given path. Then by $P[p_ip_j]$ we denote a subpath of P starting at vertex p_i and ending at vertex p_j. For a given vertex $q \in V(D)$, by q-out-branching (or q-out-tree) we denote an out-branching (out-tree) of D rooted at vertex q. We say that the removal of an arc uv (or a vertex set S) *disconnects* a vertex w from the root r if every path from r to w in D contains arc uv (or one of the

vertices in S). An arc uv is contracted as follows: add a new vertex u', and for each arc wv or wu add the arc wu' and for an arc vw or uw add the arc $u'w$, remove all arcs incident to u and v and the vertices u and v. We say that a reduction rule is *safe* for a value k if whenever the rule is applied to an instance (D, k) to obtain an instance (D', k'), D has an r-out-branching with $\geq k$ leaves if and only if D' has an r-out-branching with $\geq k'$ leaves.

Proposition 2.1. [20] *Let D be a digraph and r be a vertex from which every vertex in $V(D)$ is reachable. Then if we have an out-tree rooted at r with k leaves then we also have an out-branching rooted at r with k leaves.*

Let T be an out-tree of a digraph D. We say that u is a *parent* of v and v is a *child* of u if $uv \in A(T)$. We say that u is an *ancestor* of v if there is a directed path from u to v in T. An arc uv in $A(D) \setminus A(T)$ is called a *forward* arc if u is an ancestor of v, a *backward* arc if v is an ancestor of u and a *cross* arc, otherwise.

3. Reduction Rules for Rooted k-Leaf Out-Branching

In this section we give all the data reduction rules we apply on the given instance of Rooted k-Leaf Out-Branching to shrink its size.

Reduction Rule 1. [Reachability Rule] *If there exists a vertex u which is disconnected from the root r, then return* No.

For the Rooted k-Leaf Out-Tree problem, Rule 1 translates into the following: If a vertex u is disconnected from the root r, then remove u and all in-arcs and out-arcs of u.

Reduction Rule 2. [Useless Arc Rule] *If vertex u disconnects a vertex v from the root r, then remove the arc vu.*

Lemma 3.1. *Reduction Rules 1 and 2 are safe.*

Reduction Rule 3. [Bridge Rule] *If an arc uv disconnects at least two vertices from the root r, contract arc uv.*

Lemma 3.2. *Reduction Rule 3 is safe.*

Reduction Rule 4. [Avoidable Arc Rule] *If a vertex set S, $|S| \leq 2$, disconnects a vertex v from the root r, $vw \in A(D)$ and $xw \in A(D)$ for all $x \in S$, then delete the arc vw.*

Lemma 3.3. *Reduction Rule 4 is safe.*

Reduction Rule 5. [Two Directional Path Rule] *If there is a path $P = p_1 p_2 \dots p_{l-1} p_l$ with $l = 7$ or $l = 8$ such that*

- *p_1 and $p_{in} \in \{p_{l-1}, p_l\}$ are the only vertices with in-arcs from the outside of P.*
- *p_l and $p_{out} \in \{p_1, p_2\}$ are the only vertices with out-arcs to the outside of P.*
- *The path P is the unique out-branching of $D[V(P)]$ rooted at p_1.*
- *There is a path Q that is the unique out-branching of $D[V(P)]$ rooted at p_{in}.*
- *The vertex after p_{out} on P is not the same as the vertex after p_l on Q.*

Then delete $R = P \setminus \{p_1, p_{in}, p_{out}, p_l\}$ and all arcs incident to these vertices from D. Add two vertices u and v and the arc set $\{p_{out}u, uv, vp_{in}, p_l v, vu, up_1\}$ to D.

Notice that every vertex on P has in-degree at most 2 and out-degree at most 2. Figure 1 gives an example of an application of Reduction Rule 5.

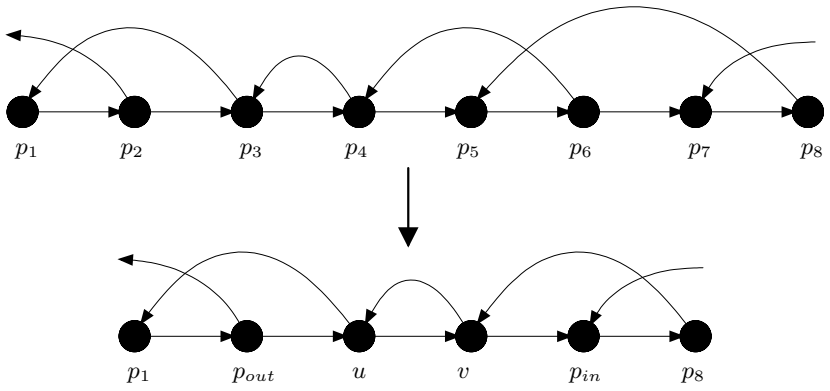


Figure 1: An Illustration of Reduction Rule 5.

Lemma 3.4. *Reduction Rule 5 is safe.*

Proof. Let D' be the graph obtained by performing Reduction Rule 5 to a path P in D. Let P_u be the path $p_1p_{out}uvp_{in}p_l$ and Q_v be the path $p_{in}p_lvup_1p_{out}$. Notice that P_u is the unique out-branching of $D'[V(P_u)]$ rooted at p_1 and that Q_v is the unique out-branching of $D'[V(P_u)]$ rooted at p_{in}.

Let T be an r-out-branching of D with at least k leaves. Notice that since P is the unique out-branching of $D[V(P)]$ rooted at p_1, Q is the unique out-branching of $D[V(P)]$ rooted at p_{in} and p_1 and p_{in} are the only vertices with in-arcs from the outside of P, $T[V(P)]$ is either a path or the union of two vertex disjoint paths. Thus, T has at most two leaves in $V(P)$ and at least one of the following three cases must apply.

(1) $T[V(P)]$ is the path P from p_1 to p_l.
(2) $T[V(P)]$ is the path Q from p_{in} to p_{out}.
(3) $T[V(P)]$ is the vertex disjoint union of a path $\tilde{P}$ that is a subpath of P rooted at p_1, and a path $\tilde{Q}$ that is a subpath of Q rooted at p_{in}.

In the first case we can replace the path P in T by the path P_u to get an r-out-branching of D' with at least k leaves. Similarly, in the second case, we can replace the path Q in T by the path Q_v to get an r-out-branching of D' with at least k leaves. For the third case, observe that $\tilde{P}$ must contain p_{out} since $p_{out} = p_1$ or p_1 appears before p_{out} on Q and thus, p_{out} can only be reached from p_1. Similarly, $\tilde{Q}$ must contain p_l. Thus, $T \setminus R$ is an r-out-branching of $D \setminus R$. We build an r-out-branching T' of D' by taking $T \setminus R$ and letting u be the child of p_{out} and v be the child of p_l. In this case T and T' have same number of leaves outside of $V(P)$ and T has at most two leaves in $V(P)$ while both u and v are leaves in T'. Hence T' has at least k leaves.

The proof for the reverse direction is similar. ∎

A digraph D is a *reduced instance* of Rooted k-Leaf Out-Branching if none of the reduction rules (Rules 1–5) can be applied to D. The following statement is easy to see:

Lemma 3.5. *For a digraph D on n vertices, we can obtain a reduced instance D' in polynomial time.*

4. Polynomial Kernel: Bounding a Reduced No-instance

Here, we show that any reduced no-instance of Rooted k-Leaf Out-Branching must have at most $O(k^3)$ vertices. In order to do so we start with T, a breadth-first search-tree (or BFS-tree for short) rooted at r, of a reduced instance D and look at a path P of T such that every vertex on P has out-degree one in T. We bound the number of endpoints of arcs with one endpoint in P and one endpoint outside of P (Section 4.1). We then use these results to bound the size of any maximal path with every vertex having out-degree one in T (Section 4.2). Finally, we combine these results to bound the size of any reduced no-instance of Rooted k-Leaf Out-Branching by $O(k^3)$.

4.1. Bounding the Number of Entry and Exit Points of a Path

Let D be a reduced no-instance, and T be a BFS-tree rooted at r. The BFS-tree T has at most $k-1$ leaves and hence at most $k-2$ vertices with out-degree at least 2 in T. Now, let $P = p_1p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T (P does not need to be a maximal path of T). Let T_1 be the subtree of T induced by the vertices reachable from r in T without using vertices in P and let T_2 be the subtree of T rooted at the child r_2 of p_l in T. Since T is a BFS-tree, it does not have any forward arcs, and thus p_lr_2 is the only arc from P to T_2. Thus all arcs originating in P and ending outside of P must have their endpoint in T_1.

Lemma 4.1. *Let D be a reduced instance, T be a BFS-tree rooted at r, and $P = p_1p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T. Let $up_i \in A(D)$, for some i between 1 and l, be an arc with $u \notin P$. There is a path P_{up_i} from r to p_i using the arc up_i, such that $V(P_{up_i}) \cap V(P) \subseteq \{p_i, p_l\}$.*

Proof. Let T_1 be the subtree of T induced by the vertices reachable from r in T without using vertices in P and let T_2 be the subtree of T rooted at the child r_2 of p_l in T. If $u \in V(T_1)$ there is a path from r to u avoiding P. Appending the arc up_i to this path yields the desired path P_{up_i}, so assume $u \in V(T_2)$. If all paths from r to u use the arc $p_{l-1}p_l$ then $p_{l-1}p_l$ is an arc disconnecting p_l and r_2 from r, contradicting the fact that Reduction Rule 3 can not be applied. Let P' be a path from r to u not using the arc $p_{l-1}p_l$. Let x be the last vertex from T_1 visited by P'. Since P' avoids $p_{l-1}p_l$ we know that P' does not visit any vertices of $P \setminus \{p_l\}$ after x. We obtain the desired path P_{up_i} by taking the path from r to x in T_1 followed by the subpath of P' from x to u appended by the arc up_i. ∎

Corollary 4.2. *Let D be a reduced no-instance, T be a BFS-tree rooted at r and $P = p_1p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T. There are at most k vertices in P that are endpoints of arcs originating outside of P.*

Lemma 4.3. *Let D be a reduced no-instance, T be a BFS-tree rooted at r and $P = p_1p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T. There are at most $7(k-1)$ vertices outside of P that are endpoints of arcs originating in P.*

Proof. Let X be the set of vertices outside P which are out-neighbors of the vertices on P. Let P' be the path from r to p_1 in T and r_2 be the unique child of p_l in T. First, observe that since there are no forward arcs, r_2 is the only out-neighbor of vertices in $V(P)$ in the subtree of T rooted at r_2. In order to bound the size of X, we differentiate between two kinds of out-neighbors of vertices on P: (a) Out-neighbors of P that are not in $V(P')$; and

(b) Out-neighbors of P in $V(P')$. First, observe that $|X \setminus V(P')| \leq k-1$. Otherwise we could have made an r-out-tree with at least k leaves by taking the path $P'P$ and adding $X \setminus V(P')$ as leaves with parents in $V(P)$.

In the rest of the proof we bound $|X \cap V(P')|$. Let Y be the set of vertices on P' with out-degree at least 2 in T and let $P_1, P_2, \ldots, P_t$ be the remaining subpaths of P' when vertices in Y are removed. For every $i \leq t$, $P_i = v_{i1}v_{i2}\ldots v_{iq}$. We define the vertex set Z to contain the two last vertices of each path P_i. The number of vertices with out-degree at least 2 in T is upper bounded by $k-2$ as T has at most $k-1$ leaves. Hence, $|Y| \leq k-2$, $t \leq k-1$ and $|Z| \leq 2(k-1)$.

Claim 1. *For every path $P_i = v_{i1}v_{i2}\ldots v_{iq}$, $1 \leq i \leq t$, there is either an arc u_iv_{iq-1} or u_iv_{iq} where $u_i \notin V(P_i)$.*

To see the claim observe that the removal of arc $v_{iq-2}v_{iq-1}$ does not disconnect the root r from both v_{iq-1} and v_{iq} else Rule 3 would have been applicable to our reduced instance. For brevity assume that v_{iq-1} is reachable from r after the removal of arc $v_{iq-2}v_{iq-1}$. Hence there exists a path from r to v_{iq}. Let u_iv_{iq} be the last arc of this path. The fact that the BFS-tree T does not have any forward arcs implies that $u_i \notin V(P_i)$.

To every path $P_i = v_{i1}v_{i2}\ldots v_{iq}$, $1 \leq i \leq t$, we associate an interval $I_i = v_{i1}v_{i2}\ldots v_{iq-2}$ and an arc $u_iv_{iq'}$, $q' \in \{q-1, q\}$. This arc exists by Claim 1. Claim 1 and Lemma 4.1 together imply that for every path P_i there is a path P_{ri} from the root r to $v_{iq'}$ that does not use any vertex in $V(P_i)\setminus\{v_{iq-1}, v_{iq}\}$ as an intermediate vertex. That is, $V(P_{ri} \cap (V(P_i) \setminus \{v_{iq-1}, v_{iq}\}) = \emptyset$.

Let P'_{ri} be a subpath of P_{ri} starting at a vertex x_i before v_{i1} on P' and ending in a vertex y_i after v_{iq-2} on P'. We say that a path P'_{ri} *covers* a vertex x if x is on the subpath of P' between x_i and y_i and we say that it *covers* an interval I_j if x_i appears before v_{j1} on the path P' and y_i appears after v_{jq-2} on P'. Hence, the path P'_{ri} covers the interval I_i.

Let $\mathcal{P} = \{P'_1, P'_2, \ldots, P'_l\} \subseteq \{P'_{r1}, \ldots, P'_{rt}\}$ be a minimum collection of paths, such that every interval I_i, $1 \leq i \leq t$, is covered by at least one of the paths in $\mathcal{P}$. Furthermore, let the paths of $\mathcal{P}$ be numbered by the appearance of their first vertex on P'. The minimality of $\mathcal{P}$ implies that for every $P'_i \in \mathcal{P}$ there is an interval $I'_i \in \{I_1, \ldots, I_t\}$ such that P'_i is the only path in $\mathcal{P}$ that covers I'_i.

Claim 2. *For every $1 \leq i \leq l$, no vertex of P' is covered by both P'_i and P'_{i+3}.*

The path P'_{i+1} is the only path in $\mathcal{P}$ that covers the interval I'_{i+1} and hence P'_i does not cover the last vertex of I'_{i+1}. Similarly P'_{i+2} is the only path in $\mathcal{P}$ that covers the interval I'_{i+2} and hence P'_{i+3} does not cover the first vertex of I'_{i+2}. Thus the set of vertices covered by both P'_i and P'_{i+3} is empty.

Since paths P'_i and P'_{i+3} do not cover a common vertex, we have that the end vertex of P'_i appears before the start vertex of P'_{i+3} on P' or is the same as the start vertex of P'_{i+3}. Partition the paths of $\mathcal{P}$ into three sets $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2$, where path $P'_i \in \mathcal{P}_{i \bmod 3}$. Also let $\mathcal{I}_i$ be the set of intervals covered by $\mathcal{P}_i$. Observe that every interval I_j, $1 \leq j \leq t$, is part of some $\mathcal{I}_i$ for $i \in \{0, 1, 2\}$.

Let $i \leq 3$ and consider an interval $I_j \in \mathcal{I}_i$. There is a path $P_{j'} \in \mathcal{P}_i$ that covers I_j such that both endpoints of $P_{j'}$ and none of the inner vertices of $P_{j'}$ lie on P'. Furthermore for any pair of paths P_a, $P_b \in \mathcal{P}_i$ such that $a < b$, there is a subpath in P' from the endpoint of P_a to the starting point of P_b. Thus for every $i \leq 3$ there is a path P^*_i from the root r to p_1 which does not use any vertex of the intervals covered by the paths in $\mathcal{P}_i$.

We now claim that the total number of vertices on intervals I_j, $1 \leq j \leq t$, which are out-neighbors of vertices on $V(P)$ is bounded by $3(k-1)$. If not, then for some i, the number of out-neighbors in $\mathcal{I}_i$ is at least k. Now we can make an r-out-tree with k leaves by taking any r-out-tree in $D[V(P_i^*) \cup V(P)]$ and adding the out-neighbors of the vertices on $V(P)$ in $\mathcal{I}_i$ as leaves with parents in $V(P)$.

Summing up the obtained upper bounds yields $|X| \leq (k-1)+|\{r_2\}|+|Y|+|Z|+3(k-1) \leq (k-1)+1+(k-2)+2(k-1)+3(k-1) = 7(k-1)$, concluding the proof. ∎

Remark: Observe that the path P used in Lemmas 4.1 and 4.3 and Corollary 4.2 need not be a maximal path in T with its vertices having out-degree one in T.

4.2. Bounding the Length of a Path: On Paths Through Nice Forests

For a reduced instance D, a BFS-tree T of D rooted at r, let $P = p_1p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T, and let S be the set of vertices in $V(P) \setminus \{p_l\}$ with an in-arc from the outside of P.

Definition 4.4. A subforest $F = (V(P), A(F))$ of $D[V(P)]$ is said to be a *nice forest of* P if the following three properties are satisfied: (a) F is a forest of directed trees rooted at vertices in S; (b) If $p_ip_j \in A(F)$ and $i < j$ then p_i has out-degree at least 2 in F or p_j has in-degree 1 in D; and (c) If $p_ip_j \in A(F)$ and $i > j$ then for all $q > i$, $p_qp_j \notin A(D)$.

In order to bound the size of a reduced no-instance D we are going to consider a nice forest with the maximum number of leaves. However, in order to do this, we first need to show the existence of a nice forest of P.

In the following discussion let D be a reduced no-instance, T be a BFS-tree T of D rooted at r, $P = p_1p_2 \dots p_l$ be a path in T such that all vertices in $V(P)$ have out-degree 1 in T and S be the set of vertices in $V(P) \setminus \{p_l\}$ with an in-arc from the outside of P.

Lemma 4.5. *There is a nice forest in* P.

For a nice forest F of P, we define the set of *key* vertices of F to be the set of vertices in S, the leaves of F, the vertices of F with out-degree at least 2 and the set of vertices whose parent in F has out-degree at least 2.

Lemma 4.6. *Let* F *be a nice forest of* P. *There are at most* $5(k-1)$ *key vertices of* F.

We can now turn our attention to a nice forest F of P with the maximum number of leaves. Our goal is to show that if the key vertices of F are too spaced out on P then some of our reduction rules must apply. We need some more observations concerning P and F.

Observation 4.7. [Unique Path] For any two vertices p_i, p_j in $V(P)$ such that $i < j$, $p_ip_{i+1} \dots p_j$ is the only path from p_i to p_j in $D[V(P)]$.

Corollary 4.8. *No arc* p_ip_{i+1} *is a forward arc of* F.

Observation 4.9. Let p_tp_j be an arc in $A(F)$ such that neither p_t nor p_j are key vertices, and $t \in \{j-1, j+1, \dots, l\}$. Then for all $q > t$, $p_qp_j \notin A(D)$.

Observation 4.9 follows directly from the definitions of a nice forest and key vertices.

Observation 4.10. If neither p_i nor p_{i+1} are key vertices, then either $p_ip_{i+1} \notin A(F)$ or $p_{i+1}p_{i+2} \notin A(F)$.

In the following discussion let F be a nice forest of P with the maximum number of leaves and let $P' = p_x p_{x+1} \dots p_y$ be a subpath of P containing no key vertices, and additionally having the property that $p_{x-1}p_x \notin A(F)$ and $p_y p_{y+1} \notin A(F)$.

Lemma 4.11. *$V(P')$ induces a directed path in F.*

In the following discussion let Q' be the directed path $F[V(P')]$.

Observation 4.12. For any pair of vertices $p_i, p_j \in V(P')$ if $i \leq j-2$ then p_j appears before p_i in Q'.

Lemma 4.13. *All arcs of $D[V(P')]$ are contained in $A(P') \cup A(F)$.*

Combining our previous observations with Lemma 4.11, we can show:

Lemma 4.14. *If $|P'| \geq 3$, there are exactly 2 vertices in P' that are endpoints of arcs starting outside of P'.*

Observation 4.15. Let $Q' = F[V(P')]$. For any pair of vertices u, v such that there is a path $Q'[uv]$ from u to v in Q', $Q'[uv]$ is the unique path from u to v in $D[V(P')]$.

Lemma 4.16. *For any vertex $x \notin V(P')$, there are at most 2 vertices in P' with arcs to x.*

This assertion follows by combining the previously derived lemmas and observations in a proof by contradition.

Corollary 4.17. *There are $\leq 14(k-1)$ vertices in P' with out-neighbors outside of P'.*

Proof. By Lemma 4.3, there are $\leq 7(k-1)$ vertices that are endpoints of arcs originating in P'. By Lemma 4.16, each such vertex is the endpoint of ≤ 2 arcs from vertices in P'. ∎

Lemma 4.18. $|P'| \leq 154(k-1) + 10$.

Proof. Assume for contradiction that $|P'| > 154(k-1)+10$ and let X be the set of vertices in P' with arcs to vertices outside of P'. By Corollary 4.17, $|X| \leq 14(k-1)$. Hence there is a subpath of P' on at least $(154(k-1)+10)/(14(k-1)+1) = 9$ vertices containing no vertices of X. By Observation 4.10 there is a subpath $P'' = p_a p_{a+1} \dots p_b$ of P' on 7 or 8 vertices such that neither $p_{a-1}p_a$ nor $p_b p_{b+1}$ are arcs of F. By Lemma 4.11 $F[V(P'')]$ is a directed path Q''. Let p_q and p_t be the first and last vertices of Q'', respectively. By Lemma 4.14 p_a and p_q are the only vertices with in-arcs from outside of P''. By Observation 4.12 $p_q \in \{p_{b-1}, p_b\}$ and $p_t \in \{p_a, p_{a+1}\}$. By the choice of P'' no vertex of P'' has an arc to a vertex outside of P'. Furthermore, since P'' is a subpath of P' and Q'' is a subpath of Q' Lemma 4.13 implies that p_b and p_t are the only vertices of P' with out-arcs to the outside of P''. By Lemma 4.7, the path P'' is the unique out-branching of $D[V(P'')]$ rooted at p_a. By Lemma 4.15, the path Q'' is the unique out-branching of $D[V(P'')]$ rooted at p_q. By Observation 4.12 p_{b-2} appears before p_{a+2} in Q'' and hence the vertex after p_b in Q'' and p_{t+1} is not the same vertex. Thus Rule 5 can be applied on P'', contradicting the fact that D is a reduced instance. ∎

Lemma 4.19. *Let D be a reduced no-instance to* Rooted k-Leaf Out-Branching. *Then $|V(D)| = O(k^3)$. More specifically, $|V(D)| \leq 1540k^3$.*

Lemma 4.19 results in a cubic kernel for Rooted k-Leaf Out-Branching as follows.

Theorem 4.20. Rooted k-Leaf Out-Branching *and* Rooted k-Leaf Out-Tree *admits a kernel of size $O(k^3)$.*

Proof. Let D be the reduced instance of ROOTED k-LEAF OUT-BRANCHING obtained in polynomial time using Lemma 3.5. If $|V(D)| > 1540k^3$, then return YES. Else, we have an instance of size bounded by $O(k^3)$. The correctness of this step follows from Lemma 4.19 which shows that any reduced no-instance to ROOTED k-LEAF OUT-BRANCHING has size bounded by $O(k^3)$. The result for ROOTED k-LEAF OUT-TREE follows similarly. ∎

5. Kernelization Lower Bounds

In the last section we gave a cubic kernel for ROOTED k-LEAF OUT-BRANCHING. It is natural to ask whether the closely related k-LEAF OUT-BRANCHING has a polynomial kernel. The answer to this question, somewhat surprisingly, is no, unless an unlikely collapse of complexity classes occurs. To show this we utilize a recent result of Bodlaender et al. [4] that states that any *compositional* parameterized problem does not have a polynomial kernel unless the polynomial hierarchy collapses to the third level.

Definition 5.1 ([4])**.** A *composition algorithm* for a parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ receives as input a sequence $((x_1, k), \ldots, (x_t, k))$, with $(x_i, k) \in \Sigma^* \times \mathbb{N}^+$ for each $1 \leq i \leq t$, uses time polynomial in $\sum_{i=1}^{t} |x_i| + k$, and outputs $(y, k') \in \Sigma^* \times \mathbb{N}^+$ with: (1) $(y, k') \in L \iff (x_i, k) \in L$ for some $1 \leq i \leq t$; (2) k' is polynomial in k. A parameterized problem is *compositional* if there is a composition algorithm for it.

Now we state the main result of [4] which we need for our purpose.

Theorem 5.2 ([4])**.** *Let L be a compositional parameterized language whose unparameterized version $\widetilde{L}$ is* NP*-complete. Unless* PH$=\Sigma_p^3$*, there is no polynomial kernel for L.*

By considering disjoint graph unions, a relatively simple composition shows:

Theorem 5.3. *k-*LEAF OUT-TREE *has no polynomial kernel unless* PH$=\Sigma_p^3$.

A *willow* graph [12] $D = (V, A_1 \cup A_2)$ is a directed graph such that $D' = (V, A_1)$ is a directed path $P = p_1 p_2 \ldots p_n$ on all vertices of D and $D'' = (V, A_2)$ is a directed acyclic graph with one vertex r of in-degree 0, such that every arc of A_2 is a backwards arc of P. p_1 is called the *bottom* vertex of the willow, p_n is called the *top* of the willow and P is called the *stem*. A *nice willow* graph $D = (V, A_1 \cup A_2)$ is a willow graph where $p_n p_{n-1}$ and $p_n p_{n-2}$ are arcs of D, neither p_{n-1} nor p_{n-2} are incident to any other arcs of A_2 and $D'' = (V, A_2)$ has a p_n-out-branching.

Observation 5.4. Let $D = (V, A_1 \cup A_2)$ be a nice willow graph. Every out-branching of D with the maximum number of leaves is rooted at the top vertex p_n.

Lemma 5.5. *k-*LEAF OUT-TREE *in nice willow graphs is NP-hard under Karp reductions.*

Theorem 5.6. *k-*LEAF OUT-BRANCHING *has no polynomial kernel unless* PH$=\Sigma_p^3$.

Proof. We prove that if k-LEAF OUT-BRANCHING has a polynomial kernel then so does k-LEAF OUT-TREE. Let (D, k) be an instance to k-LEAF OUT-TREE. For every vertex $v \in V$ we make an instance (D, v, k) to ROOTED k-LEAF OUT-TREE. Clearly, (D, k) is a yes-instance for k-LEAF OUT-TREE if and only if (D, v, k) is a yes-instance to ROOTED k-LEAF OUT-TREE for some $v \in V$. By Theorem 4.20 ROOTED k-LEAF OUT-TREE has a $O(k^3)$ kernel, so we can apply the kernelization algorithm for ROOTED k-LEAF OUT-TREE separately to each of the n instances of ROOTED k-LEAF OUT-TREE to get n instances

(D_1, v_1, k), (D_2, v_2, k), ..., (D_n, v_n, k) with $|V(D_i)| = O(k^3)$ for each $i \leq n$. By Lemma 5.5, k-Leaf Out-Branching in nice willow graphs is NP-complete under Karp reductions, so we can reduce each instance (D_i, v_i, k) of Rooted k-Leaf Out-Tree to an instance (W_i, b_i) of k-Leaf Out-Branching in nice willow graphs in polynomial time in $|D_i|$, and hence in polynomial time in k. Thus, in each such instance, $b_i \leq (k+1)^c$ for some fixed constant c independent of both n and k. Let $b_{max} = \max_{i \leq n} b_i$. Without loss of generality, $b_i = b_{max}$ for every i. This assumption is safe because if it does not hold we can modify the instance (W_i, b_i) by replacing b_i with b_{max}, subdividing the last arc of the stem $b_{max} - b_i$ times and adding an edge from r_i to each subdivision vertex.

¿From the instances (W_1, b_{max}), ..., (W_n, b_{max}) we build an instance $(D', b_{max}+1)$ of k-Leaf Out-Branching. Let r_i and s_i be the top and bottom vertices of W_i, respectively. We build D' simply by taking the disjoint union of the willows graphs $W_1, W_2, \ldots, W_n$ and adding in an arc $r_i s_{i+1}$ for $i < n$ and the arc $r_n s_1$. Let C be the directed cycle in D obtained by taking the stem of D' and adding the arc $r_n s_1$.

If for any $i \leq n$, W_i has an out-branching with at least b_{max} leaves, then W_i has an out-branching rooted at r_i with at least b_{max} leaves. We can extend this to an out-branching of D' with at least $b_{max}+1$ leaves by following C from r_i. In the other direction suppose D' has an out-branching T with at least $b_{max}+1$ leaves. Let i be the integer such that the root r of T is in $V(W_i)$. For any vertex v in $V(D')$ outside of $V(W_i)$, the only path from r to v in D' is the directed path from r to v in C. Hence, T has at most 1 leaf outside of $V(W_i)$. Thus, $T[V(W_1)]$ contains an out-tree with at least b_{max} leaves.

By assumption, k-Leaf Out-Branching has a polynomial kernel. Hence, we can apply a kernelization algorithm to get an instance (D'', k'') of k-Leaf Out-Branching with $|V(D'')| \leq (b_{max}+1)^{c_2}$ for a constant c_2 independent of n and b_{max} such that (D'', k'') is a yes-instance iff (D', b_{max}) is. Since k-Leaf Out-Tree is NP-complete, we can reduce (D'', k'') to an instance (D^*, k^*) of k-Leaf Out-Tree in polynomial time. Hence, $k^* \leq |V(D^*)| \leq (|V(D'')|+1)^{c_3} \leq (k+1)^{c_4}$ for some constants c_3 and c_4. Hence, if k-Leaf Out-Branching has a polynomial kernel then so does k-Leaf Out-Tree. Theorem 5.3 implies that k-Leaf Out-Branching has no polynomial kernel unless PH=Σ_p^3. ∎

6. Conclusion and Discussions

We demonstrated that Turing kernelization is a more poweful technique than many-to-one kernelization. We showed that while k-Leaf Out-Branching and k-Leaf Out-Tree do not have a polynomial kernel unless an unlikely collapse of complexity classes occurs, they do have n independent cubic kernels. Our paper raises far more questions than it answers. We believe that there are many more problems waiting to be addressed from the viewpoint of Turing kernelization. A few concrete open problems in this direction are as follows:

(1) Is there a framework to rule out the possibility of $|I|^{O(1)}$ polynomial kernels similar to the framework developed in [4]?
(2) Which other problems admit a Turing kernelization like the cubic kernels for k-Leaf Out-Branching and k-Leaf Out-Tree obtained here?
(3) Does there exist a problem for which we do not have a linear many-to-one kernel, but does have linear kernels from the viewpoint of Turing kernelization?

References

[1] N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, S. Saurabh. *Parameterized algorithms for Directed Maximum Leaf problems.* ICALP, LNCS 4596, 352–362 (2007).

[2] N. Alon, F. V. Fomin, G. Gutin, M. Krivelevich, S. Saurabh. *Better algorithms and bounds for Directed Maximum Leaf problems.* FSTTCS, LNCS 4855, 316–327 (2007).

[3] H. L. Bodlaender, E. D. Demaine, M. R. Fellows, J. Guo, D. Hermelin, D. Lokshtanov, M. Müller, V. Raman, J. van Rooij, F. A. Rosamond. *Open problems in parameterized and exact computation – IWPEC 2008.* Technical Report UU-CS-2008-017, Dept. of Inform. and Comput. Sci., Utrecht Univ.

[4] H. L. Bodlaender, R. G. Downey, M. R. Fellows, D. Hermelin. *On problems without polynomial kernels.* ICALP, LNCS 5125, 563–574 (2008).

[5] P. S. Bonsma, T. Brueggermann, G. J. Woeginger. *A faster FPT algorithm for finding spanning trees with many leaves.* MFCS, LNCS 2747, 259–268 (2003).

[6] P. S. Bonsma and F. Dorn. *Tight bounds and faster algorithms for Directed Max-Leaf Spanning Tree.* ESA, LNCS 5193, 222–233 (2008).

[7] Y. Caro, D. B. West, R. Yuster. *Connected domination and spanning trees with many leaves.* SIAM J. Discrete Math. 13, 202–211 (2000).

[8] J. Chen, H. Fernau, I. A. Kanj, G. Xia. *Parametric duality and kernelization: lower bounds and upper bounds on kernel size.* SIAM J. Comput. 37, 1077–1106 (2007).

[9] J. Chen, I. A. Kanj, W. Jia. *Vertex Cover: further observations and further improvements.* J. Algorithms 41, 280–301 (2001).

[10] J. Daligaut, G. Gutin, E. Jung Kim, A. Yeo. FPT algorithms and kernels for the Directed k-Leaf Problem. Technical Report 0810.4946v2, ArXiv, 2008.

[11] G. Ding, Th. Johnson, P. Seymour. *Spanning trees with many leaves.* J. Graph Theory 37, 189–197 (2001).

[12] M. Drescher and A. Vetta. *An approximation algorithm for the maximum leaf spanning arborescence problem.* To appear in ACM Trans. Algorithms.

[13] V. Estivill-Castro, M. R. Fellows, M. A. Langston, F. A. Rosamond, *FPT is P-Time extremal structure I.* Proc. ACiD, 1–41, (2005).

[14] M. R. Fellows. *The lost continent of polynomial time: preprocessing and kernelization.* IWPEC, LNCS 4169, 276–277 (2006).

[15] M. R. Fellows, C. McCartin, F. A. Rosamond, U. Stege. *Coordinated kernels and catalytic reductions: an improved FPT algorithm for Max Leaf Spanning Tree and other problems.* FSSTCS, LNCS 1974, 240–251 (2000).

[16] G. Galbiati, A. Morzenti, F. Maffioli. *On the approximability of some maximum spanning tree problems.* Theoret. Comp. Sci. 181, 107–118 (1997).

[17] J. Guo and R. Niedermeier. *Invitation to data reduction and problem kernelization.* ACM SIGACT News. 38, 31–45 (2007).

[18] R. Karp. *Reducibility among combinatorial problems.* Complexity of Computer Computations (Symposium Proceedings), Plenum Press (1972).

[19] D. J. Kleitman and D. B. West. *Spanning trees with many leaves.* SIAM J. Discrete Mathematics 4, 99–106 (1991).

[20] J. Kneis, A. Langer and P. Rossmanith. *A new algorithm for finding trees with many leaves.* To appear in Proc. of ISAAC 2008.

[21] N. Linial and D. Sturtevant. Storer's lower bound [24] holds even for graphs with minimum degree three. (quoted in [19])

[22] H. I. Lu and R. Ravi. *Approximating maximum leaf spanning trees in almost linear time.* J. Algorithms 29, 132–141 (1998).

[23] R. Solis-Oba. *2-approximation algorithm for finding a spanning tree with the maximum number of leaves.* ESA, LNCS 1461, 441–452 (1998).

[24] J. A. Storer. *Constructing full spanning trees for cubic graphs.* Inf. Process. Lett. 13, 8–11, 1981.

[25] S. Thomassé. *A quadratic kernel for feedback vertex set.* To appear in Proc. of SODA 2009.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 433–444
www.stacs-conf.org

FORWARD ANALYSIS FOR WSTS, PART I: COMPLETIONS

ALAIN FINKEL [1] AND JEAN GOUBAULT-LARRECQ [1,2]

[1] LSV, ENS Cachan, CNRS; 61 avenue du président Wilson, F-94230 Cachan

[2] INRIA Saclay Ile-de-France
E-mail address: {finkel,goubault}@lsv.ens-cachan.fr

ABSTRACT. Well-structured transition systems provide the right foundation to compute a finite basis of the set of predecessors of the upward closure of a state. The dual problem, to compute a finite representation of the set of successors of the downward closure of a state, is harder: Until now, the theoretical framework for manipulating downward-closed sets was missing. We answer this problem, using insights from domain theory (dcpos and ideal completions), from topology (sobrifications), and shed new light on the notion of adequate domains of limits.

1. Introduction

The theory of well-structured transition systems (WSTS) is 20 years old [9, 11, 2]. The most often used result of this theory [11] is the backward algorithm for computing a finite basis of the set $\uparrow Pre^*(\uparrow s)$ of predecessors of the upward closure $\uparrow s$ of a state s. The starting point of this paper is our desire to compute $\downarrow Post^*(\downarrow s)$ in a similar way. We then need a theory to finitely (and effectively) represent downward-closed sets, much as upward-closed subsets can be represented by their finite sets of minimal elements. This will serve as a basis for constructing forward procedures.

The *cover*, $\downarrow Post^*(\downarrow s)$, contains more information than the set of predecessors $\uparrow Pre^*(\uparrow s)$ because it characterizes a good approximation of the reachability set, while the set of predecessors describes the states from which the system may fail; the cover may also allow the computation of a finite-state abstraction of the system as a symbolic graph. Moreover, the backward algorithm needs a finite basis of the upward closed set of bad states, and its implementation is, in general, less efficient than a forward procedure: e.g., for lossy channel systems, although the backward procedure always terminates, only the non-terminating forward procedure is implemented in the tool TREX [1].

Except for some partial results [9, 7, 13], a general theory of downward-closed sets is missing. This may explain the scarcity of forward algorithms for WSTS. Quoting Abdulla *et al.* [3]: "Finally, we aim at developing generic methods for building downward closed languages, in a similar manner to the methods we have developed for building upward closed languages in [2]. This would give a general theory for forward analysis of infinite state systems, in the same way the work in [2] is for backward analysis." Our contribution is to provide such a theory of downward-closed sets.

Key words and phrases: WSTS, forward analysis, completion, Karp-Miller procedure, domain theory, sober spaces, Noetherian spaces.

Related Work. Karp and Miller [16] proposed an algorithm that computes a finite representation of the downward closure of the reachability set of a Petri net. Finkel [9] introduced the WSTS framework and generalized the Karp-Miller procedure to a class of WSTS. This is done by constructing the completion of the set of states (by ideals, see Section 3) and in replacing the ω-acceleration of an increasing sequence of states (in Petri nets) by its least upper bound (lub). However, there are no effective finite representations of downward closed sets in [9]. Emerson and Namjoshi [7] considered a variant of WSTS (using cpos, but still without a theory of effective finite representations of downward-closed subsets) for defining a Karp-Miller procedure to broadcast protocols—termination is then not guaranteed [8]. Abdulla *et al.* [1] proposed a forward procedure for lossy channel systems using downward-closed languages, coded as SREs. Ganty, Geeraerts, and others [13, 12] proposed a forward procedure for solving the coverability problem for WSTS equipped with an effective adequate domain of limits. This domain ensures that every downward closed set has a finite representation; but no insight is given how these domains can be found or constructed. They applied this to Petri nets and lossy channel systems. Abdulla *et al.* [3] proposed another symbolic framework for dealing with downward closed sets for timed Petri nets.

We shall see that these constructions are special cases of our completions (Section 3). We shall illustrate this in Section 4, and generalize to a comprehensive hierarchy of data types in Section 5. We briefly touch the question of computing approximations of the cover in Section 6, although we shall postpone most of it to future work. We conclude in Section 7.

2. Preliminaries

We shall borrow from theories of order, both from the theory of well quasi-orderings, as used classically in well-structured transition systems [2, 11], and from domain theory [5, 14]. We should warn the reader that this is one bulky section on preliminaries. We invite her to skip technical points first, returning to them on demand.

A *quasi-ordering* $\leq$ is a reflexive and transitive relation on a set X. It is a (partial) *ordering* iff it is antisymmetric. A set X equipped with a partial ordering is a *poset*.

We write $\geq$ the converse quasi-ordering, $\approx$ the equivalence relation $\leq \cap \geq$, $<$ associated strict ordering ($\leq \setminus \approx$), and $>$ the converse ($\geq \setminus \approx$) of $<$. The *upward closure* $\uparrow E$ of a set E is $\{y \in X \mid \exists x \in E \cdot x \leq y\}$. The *downward closure* $\downarrow E$ is $\{y \in X \mid \exists x \in E \cdot y \leq x\}$. A subset E of X is *upward closed* if and only if $E = \uparrow E$, i.e., any element greater than or equal to some element in E is again in E. *Downward closed* sets are defined similarly. When the ambient space X is not clear from context, we shall write $\downarrow_X E$, $\uparrow_X E$ instead of $\downarrow E$, $\uparrow E$.

A quasi-ordering is *well-founded* iff it has no infinite strictly descending chain, i.e., $x_0 > x_1 > \ldots > x_i > \ldots$. An *antichain* is a set of pairwise incomparable elements. A quasi-ordering is *well* if and only it is well-founded and has no infinite antichain.

There are a number of equivalent definitions for well quasi-orderings (wqo). One is that, from any infinite sequence $x_0, x_1, \ldots, x_i, \ldots$, one can extract an infinite ascending chain $x_{i_0} \leq x_{i_1} \leq \ldots \leq x_{i_k} \leq \ldots$, with $i_0 < i_1 < \ldots < i_k < \ldots$. Another one is that any upward closed subset can be written $\uparrow E$, with E *finite*. Yet another, topological definition [15, Proposition 3.1] is to say that X, with its Alexandroff topology, is Noetherian. The *Alexandroff topology* on X is that whose opens are exactly the upward closed subsets. A subset K is compact if it satisfies the Heine-Borel property, i.e., every one may extract a finite subcover from any open cover of K. A topology is *Noetherian* iff every open subset is compact, iff any increasing chain of opens stabilizes [15, Proposition 3.2]. We shall cite results from the latter paper as the need evolves.

We shall be interested in rather particular topological spaces, whose topology arises from order. A *directed family* of X is any non-empty family $(x_i)_{i\in I}$ such that, for all $i, j \in I$, there is a $k \in I$ with $x_i, x_j \leq x_k$. The *Scott topology* on X has as opens all upward closed subsets U such that every directed family $(x_i)_{i\in I}$ that has a least upper bound x in X intersects U, i.e., $x_i \in U$ for some $i \in I$. The Scott topology is coarser than the Alexandroff topology, i.e., every Scott-open is Alexandroff-open (upward closed); the converse fails in general. The Scott topology is particularly interesting on *dcpos*, i.e., posets X in which every directed family $(x_i)_{i\in I}$ has a least upper bound $\sup_{i\in I} x_i$.

The *way below* relation $\ll$ on a poset X is defined by $x \ll y$ iff, for every directed family $(z_i)_{i\in I}$ that has a least upper bound $z \geq y$, then $z_i \geq x$ for some $i \in I$ already. Note that $x \ll y$ implies $x \leq y$, and that $x' \leq x \ll y \leq y'$ implies $x' \ll y'$. However, $\ll$ is not reflexive or irreflexive in general. Write $\Uparrow E = \{y \in X \mid \exists x \in E \cdot x \ll y\}$, $\Downarrow E = \{y \in X \mid \exists x \in E \cdot y \ll x\}$. X is *continuous* iff, for every $x \in X$, $\Downarrow x$ is a directed family, and has x as least upper bound. One may be more precise: A *basis* is a subset B of X such that any element $x \in X$ is the least upper bound of a directed family of elements way below x *in* B. Then X is continuous if and only if it has a basis, and in this case X itself is the largest basis. In a continuous dcpo, $\Uparrow x$ is Scott-open for all x, and every Scott-open set U is a union of such sets, viz. $U = \bigcup_{x\in U} \Uparrow x$ [5].

X is *algebraic* iff every element x is the least upper bound of the set of finite elements below x—an element y is *finite* if and only if $y \ll y$. Every algebraic poset is continuous, and has a least basis, namely its set of finite elements.

$\mathbb{N}$, with its natural ordering, is a wqo and an algebraic poset. All its elements are finite, so $x \ll y$ iff $x \leq y$. $\mathbb{N}$ is not a dcpo, since $\mathbb{N}$ itself is a directed family without a least upper bound. Any finite product of continuous posets (resp., continuous dcpos) is again continuous, and the Scott-topology on the product coincides with the product topology. Any finite product of wqos is a wqo. In particular, $\mathbb{N}^k$, for any integer k, is a wqo and a continuous poset: this is the set of configurations of Petri nets.

It is clear how to complete $\mathbb{N}$ to make it a cpo: let $\mathbb{N}_\omega$ be $\mathbb{N}$ with a new element ω such that $n \leq \omega$ for all $n \in \mathbb{N}$. Then $\mathbb{N}_\omega$ is still a wqo, and a continuous cpo, with $x \ll y$ if and only if $x \in \mathbb{N}$ and $x \leq y$. In general, completing a wqo is necessary to extend coverability tree techniques [9, 13]. Geeraerts *et al.* (op. cit.) axiomatize the kind of completions they need in the form of so-called *adequate domains of limits*. We discuss them in Section 3. For now, let us note that the second author also proposed to use another notion of completion in another context, known as *sobrification* [15]. We need to recap what this is about.

A topological space X is always equipped with a *specialization quasi-ordering*, which we shall write $\leq$ again: $x \leq y$ if and only if any open subset containing x also contains y. X is T_0 if and only if $\leq$ is a partial ordering. Given any quasi-ordering $\leq$ on a set X, both the Alexandroff and the Scott topologies admit $\leq$ as specialization quasi-ordering. In fact, the Alexandroff topology is the finest (the one with the most opens) having this property. The coarsest is called the *upper topology*; its opens are arbitrary unions of complements of sets of the form $\downarrow E$, E finite. The latter sets $\downarrow E$, with E finite, will play an important role, and we call them the *finitary closed* subsets. Note that finitary closed subsets are closed in the upper, Scott, and Alexandroff topologies, recalling that a subset is *closed* iff its complement is open. The *closure* $cl(A)$ of a subset A of X is the smallest closed subset containing A. A closed subset F is *irreducible* if and only if F is non-empty, and whenever $F \subseteq F_1 \cup F_2$ with F_1, F_2 closed, then $F \subseteq F_1$ or $F \subseteq F_2$. The finitary closed subset $\downarrow x = cl(\{x\})$ ($x \in X$) is always irreducible. A space X is *sober* iff every irreducible closed subset F is the closure of a unique point, i.e., $F = \downarrow x$ for some unique x. Any sober space is T_0, and any continuous cpo is sober in its Scott topology. Conversely, given a T_0 space X, the space $\mathcal{S}(X)$

of all irreducible closed subsets of X, equipped with upper topology of the inclusion ordering $\subseteq$, is always sober, and the map $\eta_{\mathcal{S}} : x \mapsto \uparrow x$ is a topological embedding of X inside $\mathcal{S}(X)$. $\mathcal{S}(X)$ is the *sobrification* of X, and can be thought as X together with all missing limits from X. Note in particular that a sober space is always a cpo in its specialization ordering [5, Proposition 7.2.13].

It is an enlightening exercise to check that $\mathcal{S}(\mathbb{N})$ is $\mathbb{N}_\omega$. Also, the topology on $\mathcal{S}(\mathbb{N})$ (the upper topology) coincides with that of $\mathbb{N}_\omega$ (the Scott topology). In general, X is Noetherian if and only if $\mathcal{S}(X)$ is Noetherian [15, Proposition 6.2], however the upper and Scott topologies do not always coincide [15, Section 7]. In case of ambiguity, given any poset X, we write X_a the space X with its Alexandroff topology.

Another important construction is the *Hoare powerdomain* $\mathcal{H}(X)$ of X, whose elements are the closed subsets of X, ordered by inclusion. (We do allow the empty set.) We again equip it with the corresponding upper topology.

3. Completions of Wqos

One of the central problems of our study is the definition of a *completion* of a wqo X, with all missing limits added. Typically, the Karp-Miller construction [16] works not with $\mathbb{N}^k$, but with $\mathbb{N}_\omega^k$. We examine several ways to achieve this, and argue that they are the same, up to some details.

ADLs, WADLs. We start with Geeraerts *et al.*'s axiomatization of so-called *adequate domain of limits* for well-quasi-ordered sets X [13]. No explicit constructions for such adequate domains of limits is given, and they have to be found by trial and error. Our main result, below, is that there is a unique least adequate domain of limits: the *sobrification* $\mathcal{S}(X_a)$ of X_a. (Recall that X_a is X with its Alexandroff topology.) This not only gives a concrete construction of such an adequate domain of limits, but also shows that we do not have much freedom in defining one.

An *adequate domain of limits* [13] (ADL) for a well-ordered set X is a triple $(L, \preceq, \gamma)$ where L is a set disjoint from X (the set of *limits*); (L_1) the map $\gamma : L \cup X \to \mathbb{P}(X)$ is such that $\gamma(z)$ is downward closed for all $z \in L \cup X$, and $\gamma(x) = \downarrow_X x$ for all non-limit points $x \in X$; (L_2) there is a limit point $\top \in L$ such that $\gamma(\top) = X$; (L_3) $z \preceq z'$ if and only if $\gamma(z) \subseteq \gamma(z')$; and ($\mathrm{L}_4$) for any downward closed subset D of X, there is a finite subset $E \subseteq L \cup X$ such that $\widehat{\gamma}(E) = D$. Here $\widehat{\gamma}(E) = \bigcup_{z \in E} \gamma(z)$.

Requirement (L_2) in [13] only serves to ensure that all closed subsets of $L \cup X$ can be represented as $\downarrow_{L \cup X} E$ for some finite subset E: the closed subset $L \cup X$ itself is then exactly $\downarrow_{L \cup X} \{\top\}$. However, ($\mathrm{L}_2$) is unnecessary for this, since $L \cup X$ already equals $\downarrow_{L \cup X} E$ by (L_3), where E is the finite subset of $L \cup X$ such that $\widehat{\gamma}(E) = L \cup X$ as ensured by (L_4). Accordingly, we drop requirement (L_2):

Definition 3.1 (WADL)**.** Let X be a poset. A *weak adequate domain of limits* (WADL) on X is any triple $(L, \preceq, \gamma)$ satisfying (L_1), (L_3), and (L_4).

Proposition 3.2. *Let X be a poset. Given a WADL $(L, \preceq, \gamma)$ on X, γ defines an order-isomorphism from $(L \cup X, \preceq)$ to some subset of $\mathcal{H}(X_a)$ containing $\mathcal{S}(X_a)$.*

Conversely, assume X wqo, and let Y be any subset of $\mathcal{H}(X_a)$ containing $\mathcal{S}(X_a)$. Then $(Y \setminus \eta_{\mathcal{S}}(X_a), \preceq, \gamma)$ is a weak adequate domain of limits, where γ maps each $x \in X$ to $\downarrow_X x$ and each $F \in Y \setminus \eta_{\mathcal{S}}(X_a)$ to itself; $\preceq$ is defined by requirement (L_3).

Proof. The Alexandroff-closed subsets of X are just its downward-closed subsets. So $\gamma(z)$ is in $\mathcal{H}(X_a)$ for all z, by (L_1). Let Y be the image of γ. By (L_3), γ defines an order-isomorphism of $L \cup X$ onto Y. It remains to show that Y must contain $\mathcal{S}(X_a)$. Let F be any irreducible closed

subset of X_a. By (L_4), there is a finite subset $E \subseteq L \cup X$ such that $F = \bigcup_{x \in E} \gamma(x)$. Since F is irreducible, there must be a single $x \in E$ such that $F = \gamma(x)$. So F is in Y.

Conversely, let X be wqo, $L = Y \setminus \eta_{\mathcal{S}}(X_a)$, and γ, $\preceq$ be as in the Lemma. Properties (L_1) and (L_3) hold by definition. For (L_4), note that X_a is a Noetherian space, hence $\mathcal{S}(X_a)$ is, too [15, Proposition 6.2]. However, by [15, Corollary 6.5], every closed subset of a sober Noetherian space is finitary. In particular, take any downward closed subset D of X. This is closed in X_a, hence its image $\eta_{\mathcal{S}}(D)$ by the topological embedding $\eta_{\mathcal{S}}$ is closed in $\eta_{\mathcal{S}}(X_a)$, i.e., is of the form $\eta_{\mathcal{S}}(X_a) \cap F$ for some closed subset F of $\mathcal{S}(X_a)$. Also, $D = \eta_{\mathcal{S}}^{-1}(F)$. Since $\mathcal{S}(X_a)$ is both sober and Noetherian, F is finitary, hence is the downward-closure $\downarrow_{\mathcal{S}(X)} E'$ of some finite subset E' in $\mathcal{S}(X)$. Let E be the set consisting of the (limit) elements in $E' \cap L$, and of the (non-limit) elements $x \in X$ such that $\downarrow_X x \in E'$. We obtain $\widehat{\gamma}(E) = \bigcup_{z \in E'} z$. On the other hand, $D = \eta_{\mathcal{S}}^{-1}(F) = \{x \in X \mid \downarrow x \in \downarrow_{\mathcal{S}(X)} E'\} = \{x \in X \mid \exists z \in E' \cdot \downarrow x \subseteq z\} = \bigcup_{z \in E'} z = \widehat{\gamma}(E)$. So ($L_4$) holds. ∎

I.e., up to the coding function γ, there is a unique *minimal* WADL on any given wqo X: its sobrification $\mathcal{S}(X_a)$. There is also a unique largest one: its Hoare powerdomain $\mathcal{H}(X_a)$. An adequate domain of limits in the sense of Geeraerts *et al.* [13], i.e., one that additionally satisfies (L_2) is, up to isomorphism, any subset of $\mathcal{H}(X_a)$ containing $\mathcal{S}(X_a)$ plus the special closed set X itself as top element. We contend that $\mathcal{S}(X_a)$ is, in general, the sole WADL worth considering.

Ideal completions. We have already argued that $\mathcal{S}(X)$, for any Noetherian space X, was in a sense of completion of X, adding missing limits. Another classical construction to add limits to some poset X is its *ideal completion* $Idl(X)$. The elements of the ideal completion of X are its *ideals*, i.e., its downward-closed directed families, ordered by inclusion. $Idl(X)$ can be visualized as a form of Cauchy completion of X: we add all missing limits of directed families $(x_i)_{i \in I}$ from X, by declaring these families to be their limits, equating two families when they have the same downward-closure. In $Idl(X)$, the finite elements are the elements of X; formally, the map $\eta_{Idl} : X \to Idl(X)$ that sends x to $\downarrow x$ is an embedding, and the finite elements of $Idl(X)$ are those of the form $\eta_{Idl}(x)$. It turns out that sobrification and ideal completion coincide, in a strong sense:

Proposition 3.3 ([17])**.** *For any poset X, $\mathcal{S}(X_a) = Idl(X)$.*

This is not just an isomorphism: the irreducible closed subsets of X_a are *exactly* the ideals. Note also that $Idl(X)$ is always an algebraic dcpo [5, Proposition 2.2.22, Item 4].

When X is wqo, any downward-closed subset of X is a *finite* union of ideals. So $(Idl(X) \setminus X, \subseteq, \mathrm{id})$ is a WADL on X. Proposition 3.2 and Proposition 3.3 entail this, and a bit more:

Theorem 3.4. *For any wqo X, $\mathcal{S}(X_a) = Idl(X)$ is the smallest WADL on X.*

Well-based continuous cpos. There is a natural notion of limit in dcpos: whenever $(x_i)_{i \in I}$ is a directed family, consider $\sup_{i \in I} x_i$. Starting from a wqo X, it is then natural to look at some dcpo Y that would contain X as a basis. In particular, Y would be continuous. This prompts us to define a *well-based continuous dcpo* as one that has a well-ordered basis—namely the original poset X.

This has several advantages. First, in general there are several notions of "sets of limits" of a given subset $A \subseteq Y$, but we shall see that they all coincide in continuous posets. Such sets of limits are important, because these are what we would like Karp-Miller-like procedures to compute, through acceleration techniques. Here are the possible notions. First, define $\mathrm{Lub}_Y(A)$ as the set of all least upper bounds in Y of directed families in A. Second, $\mathrm{Ind}_Y(A)$, the *inductive hull* of A in Y, is the smallest sub-dcpo of Y containing A. Finally, the (Scott-topological) closure $cl(A)$ of A. It is well-known that $cl(A)$ is the smallest *downward closed* sub-dcpo of Y containing A.

(Recall that any open is upward closed, so that any closed set must be downward closed.) In any dcpo Y, one has $A \subseteq \mathrm{Lub}_Y(A) \subseteq \mathrm{Ind}_Y(A) \subseteq cl(A)$, and all inclusions are strict in general. E.g., in $Y = \mathbb{N}_\omega$, take A to be the set of even numbers. Then $\mathrm{Lub}_Y(A) = \mathrm{Ind}_Y(A) = A \cup \{\omega\}$ while $cl(A) = \mathbb{N}_\omega$. While $\mathrm{Lub}_Y(A) = \mathrm{Ind}_Y(A)$ in this case, there are cases where $\mathrm{Lub}_Y(A)$ is itself not closed under least upper bounds of directed families, and one has to iterate the Lub_Y operator to compute $\mathrm{Ind}_Y(A)$. On continuous posets however, all these notions coincide [10, Appendix A].

Proposition 3.5. *Let Y be a continuous poset. Then, for every downward-closed subset A of Y, $\mathrm{Ind}_Y(A) = \mathrm{Lub}_Y(A) = cl(A)$.*

We shall use this in Section 6. The key point now is that, again, well-based continuous dcpos coincide with completions of the form $\mathcal{S}(X_a)$ or $Idl(X)$, and are therefore WADLs [10, Appendix B]. This even holds for continuous dcpos having a well-founded (not well-ordered) basis:

Proposition 3.6. *Any continuous dcpo Y with a well-founded basis is order-isomorphic to $Idl(X)$ for some well-ordered set X. One may take the subset of finite elements of X for Y. If Y is well-based, then X is well-ordered.*

4. Some Concrete WADLs

We now build WADLs for several concrete posets X. Following Proposition 3.2, it suffices to characterize $\mathcal{S}(X_a)$. Although $\mathcal{S}(X_a) = Idl(X)$ (Proposition 3.3), the mathematics of $\mathcal{S}(X_a)$ is easier to deal with than $Idl(X)$.

$\mathbb{N}^k$. We start with $X = \mathbb{N}^k$, with the pointwise ordering. We have already recalled from [15] that $\mathcal{S}(\mathbb{N}^k_a)$ was, up to isomorphism, $(\mathbb{N}_\omega)^k$, ordered with the pointwise ordering, where ω is a new element above any natural number. This is the structure used in the standard Karp-Miller construction for Petri nets [16].

Σ^*. Let Σ be a finite alphabet. The *divisibility ordering* $|$ on Σ^*, a.k.a. the subsequence (non-continuous subword) ordering, is defined by $a_1 a_2 \ldots a_n \mid w_0 a_1 w_1 a_2 \ldots a_n w_n$, for any letters $a_1, a_2, \ldots, a_n \in \Sigma$ and words $w_0, w_1, \ldots, w_n \in \Sigma^*$. There is a more general definition, where letters themselves are quasi-well-ordered. Our definition is the special case where the wqo on letters is $=$, and is the one required in verifying lossy channel systems [4]. Higman's Lemma states that $|$ is wqo on Σ^*.

Any upward closed subset U of Σ^* is then of the form $\uparrow E$, with E finite. For any element $w = a_1 a_2 \ldots a_n$ of E, $\uparrow w$ is the regular language $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_n \Sigma^*$. Forward analysis of lossy channel systems is instead based on simple regular expressions (SREs). Recall from [1] that an *atomic expression* is any regular expression of the form $a^?$, with $a \in \Sigma$, or A^*, where A is a non-empty subset of Σ. When $A = \{a_1, \ldots, a_m\}$, we take A^* to denote $(a_1 + \ldots + a_m)^*$; $a^?$ denotes $\{a, \epsilon\}$. A *product* is any regular expression of the form $e_1 e_2 \ldots e_n$ ($n \in \mathbb{N}$), where each e_i is an atomic expression. A *simple regular expression*, or *SRE*, is a sum, either $\emptyset$ or $P_1 + \ldots + P_k$, where $P_1, \ldots, P_k$ are products. Sum is interpreted as union. That SREs and products are relevant here is no accident, as the following proposition shows.

Proposition 4.1. *The elements of $\mathcal{S}(\Sigma^*_a)$ are exactly the denotations of products. The downward closed subsets of Σ^* are exactly the denotations of SREs.*

Proof. The second part is well-known. If $F = P_1 + \ldots + P_k$ is irreducible closed, then by irreducibility k must equal 1, hence F is denoted by a product. Conversely, it is easy to show that any product denotes an ideal, hence an element of $Idl(X) = \mathcal{S}(X_a)$ (Proposition 3.3). ∎

Inclusion between products can then be checked in quadratic time [1]. Inclusion between SREs can be checked in polynomial time, too, because of the remarkable property that $P_1 + \ldots + P_m \subseteq P'_1 + \ldots + P'_n$ if and only if, for every i ($1 \leq i \leq m$), there is a j ($1 \leq j \leq n$) with $P_i \subseteq P'_j$ [1, Lemma 1].Similar lemmas are given by Abdulla *et al.* [3, Lemma 3, Lemma 4] for more general notions of SREs on words on infinite alphabets, and for a similar notion for finite multisets of elements from a finite set (both will be special cases of our constructions of Section 5). This is again no accident, and is a general fact about Noetherian spaces:

Proposition 4.2. *Let X be a Noetherian space, e.g., a wqo with its Alexandroff topology. Every closed subset F of X is a finite union of irreducible closed subsets $C_1, \ldots, C_m$. If $C'_1, \ldots, C'_n$ are also irreducible closed, Then $C_1 \cup \ldots \cup C_m \subseteq C'_1 \cup \ldots \cup C'_n$ if and only if for every i ($1 \leq i \leq m$), there is a j ($1 \leq j \leq n$) with $C_i \subseteq C'_j$.*

Proof. For the first part, by the results of [15], $\mathcal{S}(X)$ is Noetherian and sober, which entails that F can be written $\downarrow \{x_1, \ldots, x_m\}$; now take $C_i = \eta_{\mathcal{S}}^{-1}(\downarrow x_i)$, $1 \leq i \leq m$ (see [10, Appendix C] for details). The second part is an easy consequence of irreducibility. ∎

Proposition 4.2 suggests to represent closed subsets of X as finite subsets A of $\mathcal{S}(X)$, interpreted as the closed set $\bigcup_{C \in A} C$. When $X = \Sigma_a^*$, A is a finite set of products, i.e., an SRE. When $X = \mathbb{N}_a^k$, A is a finite subset of $\mathbb{N}_\omega^k$, interpreted as $\downarrow A \cap \mathbb{N}^k$.

Finite Trees. All the examples given above are well-known. Here is one that is new, and also more involved than the previous ones. Let $\mathcal{F}$ be a finite signature of function symbols with their arities. We let $\mathcal{F}_k$ the set of function symbols of arity k; $\mathcal{F}_0$ is the set of *constants*, and is assumed to be non-empty. The set $\mathcal{T}(\mathcal{F})$ is the set of ground terms built from $\mathcal{F}$. Kruskal's Tree Theorem states that this is well-quasi-ordered by the *homeomorphic embedding* ordering $\trianglelefteq$, defined as the smallest relation such that, whenever $u = f(u_1, \ldots, u_m)$ and $v = g(v_1, \ldots, v_n)$, $u \trianglelefteq v$ if and only if $u \trianglelefteq v_j$ for some j, $1 \leq j \leq n$, or $f = g$, $m = n$, and $u_1 \trianglelefteq v_1$, $u_2 \trianglelefteq v_2$, …, $u_m \trianglelefteq v_m$. (As for Σ^*, we take a special case, where each function has fixed arity.)

The structure of $\mathcal{S}(\mathcal{T}(\mathcal{F})_a)$ is described using an extension of SREs to the tree case. This uses regular tree expressions as defined in [6, Section 2.2]. Let $\mathcal{K}$ be a countably infinite set of additional constants, called *holes* $\Box$. Most tree regular expressions are self-explanatory, except Kleene star $L^{*,\Box}$ and concatenation $L._\Box L'$. The latter denotes the set of all terms obtained from a term t in L by replacing all occurrences of $\Box$ by (possibly different) terms from L'. The language of a hole $\Box$ is just $\{\Box\}$. $L^{*,\Box}$ is the infinite union of the languages of $\Box$, L, $L._\Box L$, $L._\Box L._\Box L$, etc.

Definition 4.3 (STRE)**.** *Tree products* and *product iterators* are defined inductively by:

- Every hole $\Box$ is a tree product.
- $f^?(P_1, \ldots, P_k)$ is a tree product, for any $f \in \Sigma_k$ and any tree products $P_1, \ldots, P_k$. We take $f^?(P_1, \ldots, P_k)$ as an abbreviation for $f(P_1, \ldots, P_k) + P_1 + \ldots + P_k$.
- $(\sum_{i=1}^n C_i)^{*,\Box}._\Box P$ is a tree product, for any tree product P, any $n \geq 1$, and any product iterators C_i over $\Box$, $1 \leq i \leq n$. We write $\sum_{i=1}^n C_i$ for $C_1 + C_2 + \ldots + C_n$.
- $f(P_1, \ldots, P_k)$ is a product iterator over $\Box$ for any $f \in \Sigma_k$, where: 1. each P_i, $1 \leq i \leq k$ is either $\Box$ itself or a tree product such that $\Box$ is not in the language of P_i; and 2. $P_i = \Box$ for some i, $1 \leq i \leq k$.

A *simple tree regular expression* (STRE) is a finite sum of tree products.

A tree regular expression is *closed* iff it has no free hole, where a hole is free in $f(L_1, \ldots, L_k)$, $L_1 + \ldots + L_k$, or in $f^?(L_1, \ldots, L_k)$ iff it is free in some L_i, $1 \leq i \leq k$; the only free hole in $\Box$ is

$\Box$ itself; the free holes of $L^{*,\Box}$ are those of L, plus $\Box$; the free holes of $L._\Box L'$ are those of L', plus those of L except $\Box$. E.g., $f^?(a^?, b^?)$ and $(f(\Box, g^?(a^?)) + f(g^?(b^?), \Box))^{*,\Box}._\Box f^?(a^?, b^?)$ are closed tree products. Then [10, Appendix D]:

Theorem 4.4. *The elements of $\mathcal{S}(\mathcal{T}(\mathcal{F})_a)$ are exactly the denotations of closed tree products. The downward closed subsets of $\mathcal{T}(\mathcal{F})$ are exactly the denotations of closed STREs. Inclusion is decidable in polynomial time for tree products and for STREs.*

5. A Hierarchy of Data Types

The sobrification WADL can be computed in a compositional way, as we now show. Consider the following grammar of data types of interest in verification:

D	::=	$\mathbb{N}$	natural numbers
	|	$A_\leq$	finite set A, quasi-ordered by $\leq$
	|	$D_1 \times \ldots \times D_k$	finite product
	|	$D_1 + \ldots + D_k$	finite, disjoint sum
	|	D^*	finite words
	|	$D^\circledast$	finite multisets

By *compositional*, we mean that the sobrification of any data type D is computed in terms of the sobrifications of its arguments. E.g., $\mathcal{S}(D_a^*)$ will be expressed as some extended form of products over $\mathcal{S}(D_a)$. The semantics of data types is the intuitive one. Finite products are quasi-ordered by the pointwise quasi-ordering, finite disjoint sums by comparing elements in each summand—elements from different summands are incomparable. For any poset X (even infinite), X^* is the set of finite words over X ordered by the *embedding* quasi-ordering $\leq^*$: $w \leq^* w'$ iff, writing w as the sequence of m letters $a_1 a_2 \ldots a_m$, one can write w' as $w_0 a'_1 w_1 a'_2 w_2 \ldots w_{m-1} a'_m w'_m$ with $a_1 \leq a'_1$, $a_2 \leq a'_2$, …, $a_m \leq a'_m$. $X^\circledast$ is the set of finite multisets $\{\!|x_1, \ldots, x_n|\!\}$ of elements of X, and is quasi-ordered by $\leq^\circledast$, defined as: $\{\!|x_1, x_2, \ldots, x_m|\!\} \leq^\circledast \{\!|y_1, y_2, \ldots, y_n|\!\}$ iff there is an injective map $r : \{1, \ldots, m\} \to \{1, \ldots, n\}$ such that $x_i \leq y_{r(i)}$ for all i, $1 \leq i \leq m$. When $\leq$ is just equality, $m \leq^\circledast m'$ iff every element of m occurs at least as many times in m' as in m: this is the $\leq^m$ quasi-ordering considered, on finite sets X, by Abdulla *et al.* [3, Section 2].

The analogue of products and SREs for D^* is given by the following definition, which generalizes the Σ^* case of Section 4. Note that D is in general an *infinite* alphabet, as in [3]. The following definition should be compared with [1]. The only meaningful difference is the replacement of $(a + \epsilon)$, where a is a letter, with $C^?$, where $C \in \mathcal{S}(X_a)$. It should also be compared with the *word language generators* of [3, Section 6]. Indeed, the latter are exactly our products on $A^\circledast$, where A is a finite alphabet (in our notation, $A_\leq$, with $\leq$ given as equality).

Definition 5.1 (Product, SRE)**.** Let X be a topological space. Let X^* be the set of finite words on X. For any $A, B \subseteq X^*$, let AB be $\{ww' \mid w \in A, w' \in B\}$, A^* be the set of words on A, $A^? = A \cup \{\epsilon\}$.

Atomic expressions are either of the form $C^?$, with $C \in \mathcal{S}(X)$, or A^*, with A a non-empty finite subset of $\mathcal{S}(X)$. *Products* are finite sequences $e_1 e_2 \ldots e_k$, $k \in \mathbb{N}$, and *SREs* are finite sums of products. The denotation of atomic expressions is given by $[\![C^?]\!] = C^?$, $[\![A^*]\!] = (\bigcup_{C \in A} [\![C]\!])^*$; of products by $[\![e_1 e_2 \ldots e_k]\!] = [\![e_1]\!]\,[\![e_2]\!] \ldots [\![e_k]\!]$; of SREs by $[\![P_1 + \ldots + P_k]\!] = \bigcup_{i=1}^k [\![P_i]\!]$.

Atomic expressions are ordered by $C^? \sqsubseteq C'^?$ iff $C \subseteq C'$; $C^? \sqsubseteq A'^*$ iff $C \subseteq C'$ for some $C' \in A'$; $A^* \not\sqsubseteq C'^?$; $A^* \sqsubseteq A'^*$ iff for every $C \in A$, there is a $C' \in A'$ with $C \subseteq C'$. Products are quasi-ordered by $eP \sqsubseteq e'P'$ iff (1) $e \not\sqsubseteq e'$ and $eP \sqsubseteq P'$, or (2) $e = C^?$, $e' = C'^?$, $C \subseteq C'$ and $P \sqsubseteq P'$, or (3) $e' = A'^*$, $e \sqsubseteq A'^*$ and $P \sqsubseteq e'P'$. We let $\equiv$ be $\sqsubseteq \cap \sqsupseteq$.

Definition 5.2 (⊛-Product, ⊛-SRE)**.** Let X be a topological space. For any $A, B \subseteq X$, let $A \odot B = \{m \uplus m' \mid m \in A, m' \in B\}$, $A^{\circledast}$ be the set of multisets comprised of elements from A, $A^{\textcircled{?}} = \{\{\!|x|\!\} \mid x \in A\} \cup \{\emptyset\}$, where $\emptyset$ is the empty multiset.

The ⊛*-products* P are the expressions of the form $A^{\circledast} \odot C_1^{\textcircled{?}} \odot \ldots \odot C_n^{\textcircled{?}}$, where A is a finite subset of $\mathcal{S}(X)$, $n \in \mathbb{N}$, and $C_1, \ldots, C_n \in \mathcal{S}(X)$. Their denotation $[\![P]\!]$ is $(\bigcup_{C \in A} C)^{\circledast} \odot [\![C_1]\!]^{\textcircled{?}} \odot \ldots \odot [\![C_n]\!]^{\textcircled{?}}$. They are quasi-ordered by $P \sqsubseteq P'$, where $P = A^{\circledast} \odot C_1^{\textcircled{?}} \odot C_2^{\textcircled{?}} \odot \ldots \odot C_m^{\textcircled{?}}$ and $P' = A'^{\circledast} \odot C_1'^{\textcircled{?}} \odot C_2'^{\textcircled{?}} \odot \ldots \odot C_n'^{\textcircled{?}}$, iff: (1) for every $C \in A$, there is a $C' \in A'$ with $C \subseteq C'$, and (2) letting I be the subset of those indices i, $1 \leq i \leq m$, such that $C_i \subseteq C'$ for no $C' \in A'$, there is an injective map $r : I \to \{1, \ldots, n\}$ such that $C_i \subseteq C'_{r(i)}$ for all $i \in I$. Let $\equiv$ be $\sqsubseteq \cap \sqsupseteq$.

Theorem 5.3. *For every data type D, $\mathcal{S}(D_a)$ is Noetherian, and is computed by: $\mathcal{S}(\mathbb{N}_a) = \mathbb{N}_\omega$; $\mathcal{S}(A_{\leq_a}) = A_\leq$; $\mathcal{S}((D_1 \times \ldots \times D_k)_a) = \mathcal{S}(D_{1a}) \times \ldots \times \mathcal{S}(D_{ka})$; $\mathcal{S}((D_1 + \ldots + D_k)_a) = \mathcal{S}(D_{1a}) + \ldots + \mathcal{S}(D_{ka})$; $\mathcal{S}(D^*)$ is the set of products on D modulo $\equiv$, ordered by $\sqsubseteq$ (Definition 5.1); $\mathcal{S}(D^{\circledast})$ is the set of ⊛-products on D modulo $\equiv$, ordered by $\sqsubseteq$ (Definition 5.2).*

For any data type D, equality and ordering (inclusion) in $\mathcal{S}(D_a)$ is decidable in the polynomial hierarchy.

Proof. We show that $\mathcal{S}(D_a)$ is Noetherian and is computed as given above, by induction on the construction of D. We in fact prove the following two facts separately: (1) $\mathcal{S}(D)$ is Noetherian (D, not D_a), where D is topologized in a suitable way, and (2) $D = D_a$.

To show (1), we topologize $\mathbb{N}$ and $A_\leq$ with their Alexandroff topologies, sums and products with the sum and product topologies respectively; X^* with the *subword topology*, viz. the smallest containing the open subsets $X^*U_1X^*U_2X^* \ldots X^*U_nX^*$, $n \in \mathbb{N}$, U_1, U_2, …, U_n open in X; and $X^{\circledast}$ with the *sub-multiset topology*, namely the smallest containing the subsets $X^{\circledast} \odot U_1 \odot U_2 \odot \ldots \odot U_n$, $n \in \mathbb{N}$, where U_1, U_2, …, U_n are open subsets of X. The case of $\mathbb{N}$ has already been discussed above. When $A_\leq$ is finite, it is both Noetherian and sober. The case of finite products is by [15, Section 6], that of finite sums by [15, Section 4]. The cases of X^*, resp. $X^{\circledast}$, are dealt with in [10, Appendices E, F].

To show (2), we appeal to a series of coincidence lemmas, showing that $(X^*)_a = X_a^*$ and that $(X^{\circledast})_a = X_a^{\circledast}$ notably. The other cases are obvious.

Finally, we show that inclusion and equality are decidable in the polynomial hierarchy. For this, we show in the appendices that inclusion on $\mathcal{S}(D^*)$ is $\sqsubseteq$ on products, and is decidable by a polynomial time algorithm modulo calls to an oracle deciding inclusion in $\mathcal{S}(D)$. This is by dynamic programming. Inclusion in $\mathcal{S}(D^{\circledast})$ is $\sqsubseteq$ on ⊛-products, and is decidable by a non-deterministic polynomial time algorithm modulo a similar oracle. We conclude since the orderings on $\mathbb{N}_\omega$ and on $A_\leq$ are polynomial-time decidable, while inclusion in $\mathcal{S}(D_1 \times \ldots \times D_k) \cong \mathcal{S}(D_1) \times \ldots \times \mathcal{S}(D_k)$ and in $\mathcal{S}(D_1 + \ldots + D_k) \cong \mathcal{S}(D_1) + \ldots + \mathcal{S}(D_k)$ are polynomial time modulo oracles deciding inclusion in $\mathcal{S}(D_i)$, $1 \leq i \leq k$. ∎

Look at some special cases of this construction. First, $\mathbb{N}^k$ is the data type $\mathbb{N} \times \ldots \times \mathbb{N}$, and we retrieve that $\mathcal{S}(\mathbb{N}^k) = \mathbb{N}_\omega^k$. Second, when A is a finite alphabet, A^* is given by products, as given in the Σ^* paragraph of Section 4; i.e., we retrieve the products (and SREs) of Abdulla *et al.* [1]. The more complicated case $(A^{\circledast})^*$ was dealt with by Abdulla *et al.* [3]. We note that the elements of $\mathcal{S}((A^{\circledast})_a^*)$ are exactly their *word language generators*, which we retrieve here in a principled way. Additionally, we can deal with more complex data structures such as, e.g., $(((\mathbb{N} \times A_\leq)^* \times \mathbb{N})^{\circledast})^{\circledast}$.

Finally, note that (1) and (2) are two separate concerns in the proof of Theorem 5.3. If we are ready to relinquish orderings for the more general topological route, as advocated in [15], we could also enrich our grammar of data types with infinite constructions such as $\mathbb{P}(D)$, where $\mathbb{P}(D)$ is interpreted as the powerset of D with the so-called lower Vietoris topology. In fact, $\mathcal{S}(\mathbb{P}(X)) \cong$

$\mathcal{H}(X)$ is Noetherian whenever X is, and its elements can be represented as *finite* subsets A of $\mathcal{S}(X)$, interpreted as $\bigcup_{C\in A} C$ [10, Appendix G]. In a sense, while $\mathcal{S}(X_a) = Idl(X)$ for all ordered spaces X, the sobrification construction is more robust than the ideal completion.

6. Completing WSTS, or: Towards Forward Procedures Computing the Cover

We show how one may use our completions on wqos to deal with forward analysis of well-structured systems. We shall describe this in more detail in another paper. First note that any data type D of Section 5 is suited to applying the expand, enlarge and check algorithm [13] out of the box to this end, since then $\mathcal{S}(D_a)$ is (the least) WADL for D. We instead explore extensions of the Karp-Miller procedure [16], in the spirit of Finkel [9] or Emerson and Namjoshi [7]. While the latter assumes an already built completion, we construct it. Also, we make explicit how this kind of acceleration-based procedure really computes the cover, i.e., $\downarrow Post^*(\downarrow x)$, in Proposition 6.1.

Recall that a *well-structured transition system* (WSTS) is a triple $S = (X, \leq, (\delta_i)_{i=1}^n)$, where X is well-quasi-ordered by $\leq$, and each $\delta_i : X \to X$ is a partial monotonic transition function. (By "partial monotonic" we mean that the domain of δ_i is upward closed, and δ_i is monotonic on its domain.) Letting $Pre(A) = \bigcup_{i=1}^n \delta_i^{-1}(A)$, $Pre^0(A) = A$, and $Pre^*(A) = \bigcup_{k\in\mathbb{N}} Pre^k(A)$, it is well-known that any upward closed subset of X is of the form $\uparrow E$ for some finite $E \subseteq X$, and that $Pre^*(\uparrow E)$ is an upward-closed subset $\uparrow E'$, E' finite, that arises as $\bigcup_{k=0}^m Pre^k(\uparrow E)$ for some $m \in \mathbb{N}$. Hence, provided $\leq$ is decidable and $\delta_i^{-1}(\uparrow E)$ is computable for each finite E, it is decidable whether $x \in Pre^*(\uparrow E)$, i.e., whether one may reach $\uparrow E$ from x in finitely many steps. It is equivalent to check whether $y \in \downarrow Post^*(\downarrow x)$ for some $y \in E$, where $Post(A) = \bigcup_{i=1}^n \delta_i(A)$, $Post^0(A) = A$, and $Post^*(A) = \bigcup_{k\in\mathbb{N}} Post^k(A)$.

All the existing symbolic procedures that attempt to compute $\downarrow Post^*(\downarrow x)$, even with a finite number of accelerations (e.g., Fast, Trex, Lash), can only compute subsets of the larger set $\mathrm{Lub}(\downarrow Post^*(\downarrow x))$. In general, $\mathrm{Lub}(\downarrow Post^*(\downarrow x))$ does not admit a finite representation. On the other hand, we know that the Scott-closure $cl(Post^*(\downarrow x))$, as a closed subset of $Idl(X)$ (intersected with X itself), is always finitary. Indeed, it is also a closed subset of $\mathcal{S}(X_a)$ (Proposition 3.3), which is represented as the downward closure of finitely many elements of $\mathcal{S}(X_a)$. Since $Y = Idl(X)$ is continuous, Proposition 3.5 allows us to conclude that $\mathrm{Lub}_Y(\downarrow Post^*(\downarrow x)) = cl(Post^*(\downarrow x))$ is finitary—hence representable provided X is one of the data types of Section 5.

This leads to the following construction. Any partial monotonic map $f : X \to Y$ between quasi-ordered sets lifts to a *continuous* partial map $\mathcal{S}f : \mathcal{S}(X_a) \to \mathcal{S}(Y_a)$: for each irreducible closed subset (a.k.a., ideal) C of $\mathcal{S}(X_a)$, either $C \cap \mathrm{dom}\, f \neq \emptyset$ and $\mathcal{S}f(C) = \downarrow f(C) = \{y \in Y \mid \exists x \in C \cap \mathrm{dom}\, f \cdot y \leq f(x)\}$, or $C \cap \mathrm{dom}\, f = \emptyset$ and $\mathcal{S}f(C)$ is undefined. The *completion* of a WSTS $S = (X, \leq, (\delta_i)_{i=1}^n)$ is then the transition system $\widehat{S} = (\mathcal{S}(X_a), \subseteq, (\mathcal{S}\delta_i)_{i=1}^n)$.

For example, when $X = \mathbb{N}^k$, and S is a Petri net with transitions δ_i defined as $\delta_i(\vec{x}) = \vec{x} + \vec{d_i}$ (where $\vec{d_i} \in \mathbb{Z}^k$; this is defined whenever $\vec{x} + \vec{d} \in \mathbb{N}^k$), then $\widehat{S}$ is the transition system whose set of states is $\mathcal{S}(X) = \mathbb{N}_\omega^k$, and whose transition functions are: $\mathcal{S}\delta_i(\vec{x}) = \vec{x} + \vec{d_i}$, whenever this has only non-negative coordinates, taking the convention that $\omega + d = \omega$ for any $d \in \mathbb{Z}$.

We may emulate lossy channel systems through the following *functional-lossy* channel systems (FLCS). For simplicity, we assume just one channel and no local state; the general case would only make the presentation more obscure. An FLCS differs from an LCS in that it loses only the least amount of messages needed to enable transitions. Take $X = \Sigma^*$ for some finite alphabet Σ of messages; the transitions are either of the form $\delta_i(w) = wa_i$ for some fixed letter a_i (sending a_i onto the channel), or of the form $\delta_i(w) = w_2$ whenever w is of the form $w_1 a_i w_2$, with w_1 not containing

a_i (expecting to receive a_i). Any LCS is cover-equivalent to the FLCS with the same sends and receives, where two systems are *cover-equivalent* if and only if they have the same sets $\downarrow Post^*(F)$ for any downward-closed F. Equating $\mathcal{S}(\Sigma_a^*)$ with the set of products, as advocated in Section 4, we find that transition functions of the first kind lift to $\mathcal{S}\delta_i(P) = Pa_i^?$, while transition functions of the second kind lift to: $\mathcal{S}\delta_i(\epsilon)$ is undefined, $\mathcal{S}\delta_i(a^?P) = \mathcal{S}\delta_i(P)$ if $a_i \neq a$, $\mathcal{S}\delta_i(a_i^?P) = P$, $\mathcal{S}\delta_i(A^*P) = \mathcal{S}\delta_i(P)$ if $a_i \notin A$, $\mathcal{S}\delta_i(A^*P) = A^*P$ otherwise. This is exactly how Trex computes successors [1, Lemma 6].

In general, the results of Section 5 allow us to use any domain of datatypes D for the state space X of S. The construction $\widehat{S}$ then generalizes all previous constructions, which used to be defined specifically for each datatype.

The Karp-Miller algorithm in Petri nets, or the Trex procedure for lossy channel systems, gives information about the cover $\downarrow Post^*(\downarrow x)$. This is true of *any* completion $\widehat{S}$ as constructed above:

Proposition 6.1. *Let S be a WSTS. Let $\widehat{Post}$ be the $Post$ map of the completion $\widehat{S}$. For any closed subset F of $\mathcal{S}(X_a)$, $\widehat{Post}(F) = cl(Post(F \cap X))$, and $\widehat{Post}^*(F) = cl(Post^*(F \cap X))$. Hence, for any downward closed subset F of X, $\downarrow Post(F) = X \cap \widehat{Post}(F)$, $\downarrow Post^*(F) = X \cap \widehat{Post}^*(F)$.*

Proof. Let F be closed in $\mathcal{S}(X_a)$. $\widehat{Post}(F) = \bigcup_{i=1}^n cl(\delta_i(F)) = cl(\bigcup_{i=1}^n \delta_i(F)) = cl(Post(F))$, since closure commutes with (arbitrary) unions. We then claim that $\widehat{Post}^k(F) = cl(Post^k(F))$ for each $k \in \mathbb{N}$. This is by induction on k. The cases $k = 0, 1$ are obvious. When $k \geq 2$, we use the fact that, for any continuous partial map f: $(*)$ $cl(f(cl(A))) = cl(f(A))$. Then $\widehat{Post}^k(F) = \bigcup_{i=1}^n cl(\delta_i(\widehat{Post}^{k-1}(F))) = \bigcup_{i=1}^n cl(\delta_i(cl(Post^{k-1}(F)))) = \bigcup_{i=1}^n cl(\delta_i(Post^{k-1}(F)))$ (by $(*)$) $= cl(Post^k(F))$. Finally, $\widehat{Post}^*(F) = \bigcup_{k\in\mathbb{N}} \widehat{Post}^k(F) = \bigcup_{k\in\mathbb{N}} cl(Post^k(F)) = cl(Post^*(F))$. We conclude, since for any $A \subseteq X$, $\downarrow A$ is the closure of A in X_a; the topology of X_a is the subspace topology of that of $\mathcal{S}(X_a)$; so, writing cl for closure in $\mathcal{S}(X_a)$, $\downarrow A = X \cap cl(A)$. ∎

Writing F as the finite union $C_1 \cup \ldots \cup C_k$, where $C_1, \ldots, C_k \in \mathcal{S}(X_a)$, $\widehat{Post}(F)$ is computable as $\bigcup_{1\leq i_1,\ldots,i_n\leq k} \mathcal{S}\delta_1(C_{i_1}) \cup \ldots \cup \mathcal{S}\delta_n(C_{i_n})$, assuming $\mathcal{S}\delta_i$ computable for each i. (We take $\mathcal{S}\delta_j(C_i)$ to mean $\emptyset$ if undefined, for notational convenience.) Although $\mathcal{S}\delta_i$ may be uncomputable even when δ_i is, it is computable on most WSTS in use. This holds, for example, for Petri nets and lossy channel systems, as exemplified above.

So it is easy to compute $\downarrow Post(\downarrow x)$, as (the intersection of X with) $\widehat{Post}(\downarrow x)$. Computing $\downarrow Post^*(\downarrow x)$ (our goal) is also easily computed as $\widehat{Post}^*(\downarrow x)$ (intersected with X again), using acceleration techniques for loops. This is what the Karp-Miller construction does for Petri nets, what Trex does for lossy channel systems [1]. (We examine termination issues below.) Our framework generalizes all these procedures, using a weak acceleration assumption, whereby we assume that we can compute the least upper bound of the values of loops iterated k times, $k \in \mathbb{N}$. For any continuous partial map $g : Y \to Y$ (with open domain) on a dcpo Y, let the *iteration* $\overline{g}$ be the map of domain $\mathrm{dom}\, g$ such that $\overline{g}(y)$ is the least upper bound of $(g^k(y))_{k\in\mathbb{N}}$ if $y < g(y)$, and $g(y)$ otherwise. Let $\Delta = \{\mathcal{S}\delta_1, \ldots, \mathcal{S}\delta_n\}$, Δ^* be the set of all composites of finitely many maps from Δ. Our *acceleration assumption* is that one can compute $\overline{g}(y)$ for any $g \in \Delta^*$, $y \in \mathcal{S}(X_a)$. The following procedure then computes $\downarrow Post^*(\downarrow x)$, as (the intersection of X with) $\widehat{Post}^*(\downarrow x)$, itself represented as a finite union of elements of $\mathcal{S}(X_a)$: initially, let A be $\{x\}$; then, while $\widehat{Post}(A) \not\subseteq \downarrow A$, choose fairly $(g, a) \in \Delta^* \times A$ such that $a \in \mathrm{dom}\, g$ and add $\overline{g}(a)$ to A. If this terminates, A is a finite set whose downward closure is exactly $\downarrow Post^*(\downarrow x)$. Despite its simplicity, this is the essence of the Karp-Miller procedure, generalized to a large class of spaces X.

Termination is ensured for flat systems, i.e., systems whose control graph has no nested loop, as one only has to compute the effect of a finite number of loops. In general, the procedure terminates on *cover-flattable* systems, that is systems that are cover-equivalent to some flat system. Petri nets are cover-flattable, while, e.g., not all LCS are: recall that, in an LCS, $\downarrow Post^*(\downarrow x)$ is *always* representable as an SRE, however not effectively so.

7. Conclusion and Perspectives

We have developed the first comprehensive theory of downward-closed subsets, as required for a general understanding of forward analysis techniques of WSTS. This generalizes previous domain proposals on tuples of natural numbers, on words, on multisets, allowing for nested datatypes, and infinite alphabets. Each of these domains is effective, in the sense that each has finite presentations with a decidable ordering. We have also shown how the notion of sobrification $\mathcal{S}(X_a)$ was in a sense inevitable (Section 3), and described how this applied to compute downward closures of reachable sets of configurations in WSTS (Section 6). We plan to describe such new forward analysis algorithms, in more detail, in papers to come.

References

[1] P. A. Abdulla, A. Bouajjani, and B. Jonsson. On-the-fly analysis of systems with unbounded, lossy fifo channels. In *CAV'98*, Vancouver, Canada, 1998. Springer Verlag LNCS 1427.
[2] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2):109–127, 2000.
[3] P. A. Abdulla, J. Deneux, P. Mahata, and A. Nylén. Forward reachability analysis of timed Petri nets. In Y. Lakhnech and S. Yovine, editors, *FORMATS/FTRTFT*, pages 343–362. Springer Verlag LNCS 3253, 2004.
[4] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. In *LICS'93*, pages 160–170, 1993.
[5] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Comp. Sci.*, volume 3, pages 1–168. OUP, 1994.
[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `www.grappa.univ-lille3.fr/tata`, 2004.
[7] E. A. Emerson and K. S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *LICS'98*, pages 70–80, 1998.
[8] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *LICS'99*, pages 352–359, 1999.
[9] A. Finkel. Reduction and covering of infinite reachability trees. *Inf. Comput.*, 89(2):144–179, 1990.
[10] A. Finkel and J. Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. Research report, LSV, ENS Cachan, ENS Cachan, 61 avenue du président Wilson, 94230 Cachan, Dec. 2008. Full version.
[11] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comp. Sci.*, 256(1–2):63–92, 2001.
[12] P. Ganty, J.-F. Raskin, and L. van Begin. A complete abstract interpretation framework for coverability properties of WSTS. In *VMCAI'06*, pages 49–64. Springer Verlag LNCS 3855, 2006.
[13] G. Geeraerts, J.-F. Raskin, and L. van Begin. Expand, enlarge and check: New algorithms for the coverability problem of WSTS. *J. Comp. Sys. Sci.*, 72(1):180–203, 2006.
[14] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. Continuous lattices and domains. In *Encyc. Math. and its Applications*, volume 93. CUP, 2003.
[15] J. Goubault-Larrecq. On Noetherian spaces. In *LICS'07*, pages 453–462, 2007.
[16] R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comp. Sys. Sci.*, 3(2):147–195, 1969.
[17] M. Mislove. Algebraic posets, algebraic cpo's and models of concurrency. In *Topology and Category Theory in Computer Science*, pages 75–109. Clarendon Press, 1981.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 445–456
www.stacs-conf.org

APPROXIMATING ACYCLICITY PARAMETERS OF SPARSE HYPERGRAPHS

FEDOR V. FOMIN [1] AND PETR A. GOLOVACH [1] AND DIMITRIOS M. THILIKOS [2]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Supported by the Norwegian Research Council.

[2] Department of Mathematics, University of Athens, Panepistimioupolis, GR15784 Athens, Greece. Supported by the project "Kapodistrias" (ΑΠ 02839/28.07.2008) of the National and Kapodistrian University of Athens.

Abstract. The notions of hypertree width and generalized hypertree width were introduced by Gottlob, Leone, and Scarcello (PODS'99, PODS'01) in order to extend the concept of hypergraph acyclicity. These notions were further generalized by Grohe and Marx in SODA'06, who introduced the fractional hypertree width of a hypergraph. All these width parameters on hypergraphs are useful for extending tractability of many problems in database theory and artificial intelligence. Computing each of these width parameters is known to be an NP-hard problem. Moreover, the (generalized) hypertree width of an n-vertex hypergraph cannot be approximated within a logarithmic factor unless P=NP. In this paper, we study the approximability of (generalized, fractional) hyper treewidth of sparse hypergraphs where the criterion of sparsity reflects the sparsity of their incidence graphs. Our first step is to prove that the (generalized, fractional) hypertree width of a hypergraph is constant-factor sandwiched by the treewidth of its incidence graph, when the incidence graph belongs to some apex-minor-free graph class (the family of apex-minor-free graph classes includes planar graphs and graphs of bounded genus). This determines the combinatorial borderline above which the notion of (generalized, fractional) hypertree width becomes essentially more general than treewidth, justifying that way its functionality as a hypergraph acyclicity measure. While for more general sparse families of hypergraphs treewidth of incidence graphs and all hypertree width parameters may differ arbitrarily, there are sparse families where a constant factor approximation algorithm is possible. In particular, we give a constant factor approximation polynomial time algorithm for (generalized, fractional) hypertree width on hypergraphs whose incidence graphs belong to some H-minor-free graph class. This extends the results of Feige, Hajiaghayi, and Lee from STOC'05 on approximating treewidth of H-minor-free graphs.

1. Introduction

Many important theoretical and "real-world" problems can be expressed as constrained satisfaction problems (CSP). Among examples one can mention numerous problems from

1998 ACM Subject Classification: F.2.2,G.2.2.
Key words and phrases: Graph, hypergraph, hypertree width, treewidth.

different domains like Boolean satisfiability, temporal reasoning, graph coloring, belief maintenance, machine vision, and scheduling. Another example is the conjunctive-query containment problem, which is a fundamental problem in database query evaluation. In fact, as it was shown by Kolaitis and Vardi [19], CSP, conjunctive-query containment, and finding homomorphism for relational structures are essentially the same problem. The problem is known to be NP-hard in general [2] and polynomial time solvable for restricted class of acyclic queries [25]. Recently, in the database and constraint satisfaction communities various extensions of query (or hypergraph) acyclicity were studied. The main motivation for the quest for a suitable measure of acyclicity of a hypergraph (query, or relational structure) is the extension of polynomial time solvable cases (like acyclic hypergraphs) to more general instances. In this direction, Chekuri and Rajaraman in [3] introduced the notion of query width. Gottlob, Leone, and Scarcello [13, 14, 16] defined hypertree width and generalized hypertree width. Furthermore, Grohe and Marx [18] have introduced the most general parameter known so far, fractional hypertree width, and proved that CSP, restricted to instances of bounded fractional hypertree width, is polynomial time solvable.

Unfortunately, all known variants of hypertree width are NP-complete [12, 17]. Moreover, generalized hypertree width is NP-complete even when checking whether its value is at most 3 (see [17]). In the case of hypertree width, the problem is $W[2]$-hard when parameterized by k [12]. Both hypertree width and the generalized hypertree are hard to approximate. For example, the reduction of Gottlob et al. in [12] can be used to show that the generalized hypertree width of an n-vertex hypergraph cannot be approximated within a factor $c \log n$ for some constant $c > 0$ unless $\mathsf{P} = \mathsf{NP}$.

All these parameters for hypergraphs can be seen as generalizations of the treewidth of a graph. The treewidth is a fundamental graph parameter from Graph Minors Theory by Robertson and Seymour [22] and it has numerous algorithmic applications. It is an old open question whether the treewidth can be approximated within a constant factor and the best known approximation algorithm for treewidth is $\sqrt{\log OPT}$-approximation due to Feige et al. [9]. However, as it was shown by Feige et al. [9], the treewidth of an H-minor-free graph is constant factor approximable.

Our results. Our first result is combinatorial. We show that for a wide family of hypergraphs (those where the incidence graph excludes an apex graph as a minor – that is a graph that can become planar after removing a vertex) the fractional and generalized hypertree width of a hypergraph is bounded by a linear function of treewidth of its incidence graph. Apex-minor-free graph classes include planar and bounded genus graphs.

For hypergraphs whose incidence graphs are apex graphs the two parameters may differ arbitrarily, and this result determines the boundary where fractional hypertree width starts being essentially different from treewidth of the incidence graph. This indicates that hypertree width parameters are more useful as the adequate version of acyclicity for non-sparse instances.

Our proof is based on theorems from bidimensionality theory and a min-max (in terms of fractional hyperbrambles) characterization of fractional hypertree width. The proof essentially identifies what is the obstruction analogue of fractional hypertree width for incidence graphs.

Our second result applies further for sparse classes where the difference between (generalized, fractional) hypertree width of a hypergraph and treewidth of its incidence graph can be arbitrarily large. In particular, we give a constant factor approximation algorithm

for generalized and fractional hypertree width of hypergraphs with H-minor-free incidence graphs extending the results of Feige et al. [9] from treewidth to (generalized, fractional) hypertree width. The algorithm is based on a series of theorems based on the main decomposition theorem of the Robertson-Seymour's Graph Minor project. As a combinatorial corollary of our results, it follows that generalized hypertree width and fractional hypertree width differ within constant multiplicative factor if the incidence graph of the hypergraph does not contain a fixed graph as a minor.

Due the space restrictions some proofs are omitted here. They will appear in the journal paper, but also they can be found in our technical report[10].

2. Definitions and preliminaries

Basic definitions and properties. We consider finite undirected graphs without loops or multiple edges. The vertex set of a graph G is denoted by $V(G)$ and its edge set by $E(G)$ (or simply by V and E if it does not create confusion).

Let G be a graph. For a vertex v, we denote by $N_G(v)$ its *(open) neighborhood*, i.e. the set of vertices which are adjacent to v. The *closed neighborhood* of v, i.e. the set $N_G(v) \cup \{v\}$, is denoted by $N_G[v]$. For $U \subseteq V(G)$, we define $N_G[U] = \bigcup_{v \in U} N_G[v]$ (we may omit index if the graph under consideration is clear from the context). If $U \subseteq V(G)$ (or $u \in V(G)$) then $G - U$ (or $G - u$) is the graph obtained from G by the removal of vertices of U (vertex u correspondingly). Given an edge $e = \{x, y\}$ of a graph G, the graph G/e is obtained from G by contracting e; which is, to get G/e we identify the vertices x and y and remove all loops and replace all multiple edges by simple edges. A graph H obtained by a sequence of edge-contractions is said to be a *contraction* of G. A graph H is a *minor* of G if H is a subgraph of a contraction of G. We say that a graph G is *H-minor-free* when it does not contain H as a minor. We also say that a graph class $\mathcal{G}$ is *H-minor-free* (or, excludes H as a minor) when all its members are H-minor-free. An *apex graph* is a graph obtained from a planar graph G by adding a vertex and making it adjacent to some of the vertices of G. A graph class $\mathcal{G}$ is *apex-minor-free* if $\mathcal{G}$ excludes a fixed apex graph H as a minor. The *$(k \times k)$-grid* is the Cartesian product of two paths of lengths $k - 1$. A *surface* Σ is a compact 2-manifold (we always consider connected surfaces). Whenever we refer to a *Σ-embedded graph* G we consider a 2-cell embedding of G in Σ. To simplify notations, we do not distinguish between a vertex of G and the point of Σ used in the drawing to represent the vertex or between an edge and the line representing it. We also consider a graph G embedded in Σ as the union of the points corresponding to its vertices and edges. That way, a subgraph H of G can be seen as a graph H, where $H \subseteq G$. Recall that $\Delta \subseteq \Sigma$ is a (closed) disc if it is homeomorphic to $\{(x, y) : x^2 + y^2 \leq 1\}$. The *Euler genus* of a nonorientable surface Σ is equal to the nonorientable genus $\tilde{g}(\Sigma)$ (or the crosscap number). The *Euler genus* of an orientable surface Σ is $2g(\Sigma)$, where $g(\Sigma)$ is the orientable genus of Σ.

If $X \subseteq 2^A$ for some set A, then by $\bigcup X$ we denote the union of all elements of X. Recall that a *hypergraph* $\mathcal{H}$ is a pair $\mathcal{H} = (V(\mathcal{H}), E(\mathcal{H}))$ where $V(\mathcal{H})$ is a finite nonempty set of vertices, and $E(\mathcal{H})$ is a set of nonempty subsets of $V(\mathcal{H})$ called *hyperedges*, $\bigcup E(\mathcal{H}) = V(\mathcal{H})$. We consider here only hypergraphs without isolated vertices (i.e. every vertex is in some hyperedge). For vertex $v \in V(\mathcal{H})$, we denote by $E_{\mathcal{H}}(v)$ the set of its incident hyperedges. The *incidence graph* of the hypergraph $\mathcal{H}$ is the bipartite graph $I(\mathcal{H})$ with vertex set $V(\mathcal{H}) \cup E(\mathcal{H})$ such that $v \in V(\mathcal{H})$ and $e \in E(\mathcal{H})$ are adjacent in $I(\mathcal{H})$ if and only if $v \in e$.

Treewidth of graphs and hypergraphs. A *tree decomposition* of a hypergraph $\mathcal{H}$ is a pair (T, χ), where T is a tree and $\chi\colon V(T) \to 2^{V(\mathcal{H})}$ is a function associating a set of vertices $\chi(t) \subseteq V(\mathcal{H})$ (called a *bag*) to each node t of the decomposition tree T such that i) $V(\mathcal{H}) = \bigcup_{t \in V(T)} \chi(t)$, ii) for each $e \in E(\mathcal{H})$, there is a node $t \in V(T)$ such that $e \subseteq \chi(t)$, and iii) for each $v \in V(G)$, the set $\{t \in V(T)\colon v \in \chi(t)\}$ forms a subtree of T.

The *width* of a tree decomposition equals $\max\{|\chi(t)| - 1\colon t \in V(T)\}$. The *treewidth* of a hypergraph $\mathcal{H}$ is the minimum width over all tree decompositions of $\mathcal{H}$. We use notation $\mathbf{tw}(\mathcal{H})$ for the treewidth of a hypergraph $\mathcal{H}$.

It is easy to verify that for any hypergraph $\mathcal{H}$, $\mathbf{tw}(\mathcal{H}) + 1 \geq \mathbf{tw}(I(H))$. However, these parameters can differ considerably on hypergraphs. For example, for the n-vertex hypergraph $\mathcal{H}$ with one hyperedge which contains all vertices, $\mathbf{tw}(\mathcal{H}) = n - 1$ and $\mathbf{tw}(I(\mathcal{H})) = 1$.

Since $\mathbf{tw}(\mathcal{H}) \geq |e|$ for every $e \in E(\mathcal{H})$, we have that the presence of a large hyperedge results in a large treewidth of the hypergraph. The paradigm shift in the transition from treewidth to hypertree width consists in counting the covering hyperedges rather than counting the number of vertices in a bag. This parameter seems to be more appropriate, especially with respect to constraint satisfaction problems. We start with the introduction of even more general parameter of fractional hypertree width.

Hypertree width and its generalizations. In general, given a finite set A, we use the term *labeling of* A for any function $\gamma : A \to [0, 1]$. We also use the notation $\mathscr{G}(A)$ for the collection of all labellings of a set $\mathcal{A}$. The *size* of a labelling of A is defined as $|\gamma| = \sum_{x \in A} \gamma(x)$. If the values of a labelling γ are restricted to be 0 or 1, then we say that γ is a *binary* labelling of A. Clearly, the size of a binary labelling is equal to the number of the elements of A that are labelled by 1. Given a hyperedge labelling γ of a hypergraph $\mathcal{H}$, we define the set of vertices of $\mathcal{H}$ that are *blocked* by γ as

$$B(\gamma) = \{v \in V(\mathcal{H}) \mid \sum_{e \in E_{\mathcal{H}}(v)} \gamma(e) \geq 1\},$$

i.e. the set of vertices that are incident to hyperedges whose total labelling sums up to 1 or more.

A *fractional hypertree decomposition* [18] of $\mathcal{H}$ is a triple (T, χ, λ), where (T, χ) is a tree decomposition of $\mathcal{H}$ and $\lambda\colon V(T) \to \mathscr{G}(E(\mathcal{H}))$ is a function, assigning a hyperedge labeling to each node of T, such that for every $t \in V(T)$, $\chi(t) \subseteq B(\lambda(t))$, i.e. all vertices of the bag $\chi(t)$ are blocked by the labelling $\lambda(t)$. The *width* of a fractional hypertree decomposition (T, χ, λ) is $\min\{|\lambda(t)|\colon t \in V(T)\}$, and the *fractional hypertree width* $\mathbf{fhw}(\mathcal{H})$ of $\mathcal{H}$ is the minimum of the widths of all fractional hypertree decompositions of $\mathcal{H}$. If λ assigns a binary hyperedge labeling to each node of T, then (T, χ, λ) is a *generalized hypertree decomposition* [15]. Correspondingly, the *generalized hypertree width* $\mathbf{ghw}(\mathcal{H})$ of $\mathcal{H}$ is the minimum of the widths of all generalized hypertree decompositions of $\mathcal{H}$. Clearly, $\mathbf{fhw}(\mathcal{H}) \leq \mathbf{ghw}(\mathcal{H})$ but, as it was shown in [18], there are families of hypergraphs of bounded fractional hypertree width but unbounded generalized hypertree width. Notice that computing the fractional hypertree width is an NP-complete problem even for sparse graphs. To see this, take a connected graph G that is not a tree and construct a new graph H by replacing every edge of G by $|V(G)| + 1$ paths of length 2. It is easy to check that $\mathbf{tw}(G) + 1 = \mathbf{fhw}(H)$.

The proof of the next lemma follows from results of [3] about query width. For completeness, a direct proof is given in [10].

Lemma 2.1. *For any hypergraph* $\mathcal{H}$, $\mathbf{fhw}(\mathcal{H}) \leq \mathbf{ghw}(\mathcal{H}) \leq \mathbf{tw}(I(\mathcal{H})) + 1$. [Proof in [10]]

It is necessary to remark here that the fractional hypertree width of a hypergraph can be arbitrarily smaller that the treewidth of its incidence graph. Suppose that a hypergraph $\mathcal{H}'$ is obtained from the hypergraph $\mathcal{H}$ by adding a hyperedge which includes all vertices. Then $\mathbf{fhw}(\mathcal{H}') = 1$ and $\mathbf{tw}(I(\mathcal{H}')) + 1 \geq \mathbf{tw}(I(\mathcal{H})) + 1 \geq \mathbf{fhw}(\mathcal{H})$.

Hyperbrambles. Let $\mathcal{H}$ be a hypergraph. Two sets $X, Y \subseteq V(\mathcal{H})$ *touch* if $X \cap Y \neq \emptyset$ or there exists $e \in E(\mathcal{H})$ such that $e \cap X \neq \emptyset$ and $e \cap Y \neq \emptyset$. A *hyperbramble* of $\mathcal{H}$ is a set $\mathcal{B}$ of pairwise touching connected subsets of $V(\mathcal{H})$ [1]. We say that a labelling γ of $E(\mathcal{H})$ *covers* a vertex set $S \subseteq V(\mathcal{H})$ if some of its vertices are blocked by γ. The *fractional order* of a hyperbramble is the minimum k for which there is a labeling γ of size at most k covering all elements in $\mathcal{B}$. The *fractional hyperbramble number*, $\mathbf{fbn}(\mathcal{H})$, of $\mathcal{H}$ is the maximum of the fractional orders of all hyperbrambles of $\mathcal{H}$. Using [18, Theorem 11], we can prove the following lemma.

Lemma 2.2. *For any hypergraph $\mathcal{H}$, $\mathbf{fbn}(\mathcal{H}) \leq \mathbf{fhw}(\mathcal{H})$.* [Proof in [10]]

i-brambles. An *i-labeled graph* G is a triple (G, N, M) where $N, M \subseteq V(G)$, $N \cup M = V(G)$, $M - N$ and $N - M$ are independent sets of G, and for any $v \in V(G)$ its closed neighborhood $N_G[v]$ is intersecting both N and M. Notice that $\{N, M\}$ is not necessarily a partition of $V(G)$. The incidence graph $I(\mathcal{H})$ of a hypergraph $\mathcal{H}$ can be seen as an i-labeled graph $(I(\mathcal{H}), N, M)$ where $N = V(\mathcal{H})$, $M = E(\mathcal{H})$.

The result of the contraction of an edge $e = \{x, y\}$ of an i-labeled graph (G, N, M) to a vertex v_e is the i-labeled graph (G', N', M') where i) $G' = G/e$ ii) N' contains all vertices of $N - \{x, y\}$ and also the vertex v_e, in case $\{x, y\} \cap N \neq \emptyset$ and iii) M' contains all vertices of $M - \{x, y\}$ and also the vertex v_e, in case $\{x, y\} \cap M \neq \emptyset$. An i-labeled graph (G', N', M') is a *contraction* of an i-labeled graph (G, N, M) if (G', N', M') can be obtained after applying a (possibly empty) sequence of contractions to (G, N, M). The following lemma is a direct consequence of the definitions.

Lemma 2.3. *Let (G, N, M) be an i-labeled graph and let G' be a contraction of G. Then there are $N', M' \subseteq V(G')$ such that the i-labeled graph (G', N', M') is a contraction of (G, N, M).*

Let (G, N, M) be an i-labeled graph. We say that a set $S \subseteq N$ is *i-connected* if any pair $x, y \in S$ is connected by a path in $G[S \cup M]$. We say that two subsets $S, R \subseteq N$ *i-touch* either if i) $S \cap R \neq \emptyset$, or ii) there is an edge $\{x, y\}$ with $x \in S$ and $y \in R$, or iii) there is a vertex $z \in M$ such that $N_G[z]$ intersects both S and R.

Given an i-labeled graph (G, N, M) we define an i-bramble of (G, N, M) as any collection $\mathcal{B}$ of i-touching i-connected sets of vertices in N. We say that a labeling γ of M *blocks* a vertex $x \in N$ if $\sum_{y \in N_G[x] \cap M} \gamma(y) \geq 1$. We say that γ *fractionally covers* a vertex set $S \subseteq N$ if some of its vertices is blocked by γ. The *order* of an i-bramble is the minimum k for which there is a labeling γ of M of size at most k that fractionally covers all sets of $\mathcal{B}$.

The *fractional i-bramble* number $\mathbf{fibn}(G, N, M)$ of an i-labeled graph (G, N, M) is the maximum order of all i-brambles of it.

The following statement follows immediately from the definitions of hyperbrambles and i-brambles.

Lemma 2.4. *For any hypergraph $\mathcal{H}$, $\mathbf{fibn}(I(\mathcal{H}), V(\mathcal{H}), E(\mathcal{H})) = \mathbf{fbn}(\mathcal{H})$.*

Also it can be easily seen that the fractional i-bramble number is a contraction-closed parameter.

Lemma 2.5. *If an i-labeled graph (G', N', M') is the contraction of an i-labeled graph (G, N, M) then* $\mathbf{fibn}(G', N', M') \leq \mathbf{fibn}(G, N, M)$.

Obviously, i-bramble number is not a subgraph-closed parameter (not even for induced subgraphs), but we can note the following useful claim.

Lemma 2.6. *Let (G, N, M) be an i-labeled graph and $X \subseteq V(G)$ such that $G - X$ has no isolated vertices, and for every $v \in X \cap M$, $N_G[v] \subseteq X$. Then $(G - X, N - X, M - X)$ is an i-labeled graph and* $\mathbf{fibn}(G - X, N - X, M - X) \leq \mathbf{fibn}(G, N, M)$. [Proof in [10]]

3. When hypertree width is sandwiched by treewidth

Influence and valency of i-brambles. Let (G, N, M) be an i-labelled graph and $\mathcal{B}$ an i-bramble of it. We define the *influence* of $\mathcal{B}$, as $\mathbf{ifl}(\mathcal{B}) = \max_{v \in \cup\mathcal{B}} |\{x \in \cup\mathcal{B} \mid \mathbf{dist}_G(v, x) \leq 2\}|$. We also define the *valency* of $\mathcal{B}$ as the quantity $\mathbf{val}(\mathcal{B}) = \max_{v \in \cup\mathcal{B}} |\{S \in \mathcal{B} \mid v \in S\}|$.

Lemma 3.1. *If $\mathcal{B}$ is an i-bramble of an i-labeled graph (G, N, M), then the order of $\mathcal{B}$ is at least* $\frac{|\mathcal{B}|}{\mathbf{ifl}(\mathcal{B}) \cdot \mathbf{val}(\mathcal{B})}$. [Proof in [10]]

Triangulated grids. A *partially triangulated $(k \times k)$-grid* is a graph G that is obtained from a $(k \times k)$-grid (we refer to it as its *underlying grid*) after adding some edges without harming the planarity of the resulting graph. Each vertex of G will be denoted by a pair (i, j) corresponding to its coordinates in the underlying grid. We will also denote as $U(G)$ the vertices, we call them *non-marginal*, of G that in the underlying grid have degree 4 and we call the vertices in $V(G) - U(G)$ *marginal*.

Lemma 3.2. *Let (G, N, M) be an i-labeled graph, where G is a partially triangulated $(k \times k)$-grid for $k \geq 4$. Then* $\mathbf{fibn}(G, N, M) \geq k/50 - c$, *for some constant $c \geq 0$.*

Proof. We use notation $C_{i,j}$ for the set vertices of $N \cap U(G)$ that belong to the i-th row or the j-th column of the underlying grid of G. We claim that $\mathcal{B} = \{C_{i,j} \mid 2 \leq i, j \leq k-1\}$ is an i-bramble of G of order $\geq k/50 - c$, for some constant $c \geq 0$. Since $k \geq 4$, we have that each set $C_{i,j}$ is non-empty and i-connected. Notice also that the intersection of the i-th row and the j'-th column of the underlying grid of G is either a vertex in N and $C_{i,j} \cap C_{i',j'} \neq \emptyset$, or a vertex in $M - N$, but then all neighbors of it in G belong to N. Therefore, all $C_{i,j}$ and $C_{i',j'}$ should i-touch, and $\mathcal{B}$ is an i-bramble. Each vertex $v = (i, j)$ in $N(\bigcup \mathcal{B})$ is contained in exactly $2k - 5$ sets of $\mathcal{B}$ (that is $k - 2$ sets $C_{i',j'}$ that agree on the first coordinate plus $k - 2$ sets $C_{i',j'}$ that agree on the second, minus one set $C_{i,j}$ that agrees on both), therefore $\mathbf{val}(\mathcal{B}) = 2k - 5$. For each non-marginal vertex x in G, there are at most 25 non-marginal vertices within distance ≤ 2 in G (in the worst case, consider a triangulated (5×5)-grid subgraph of G that is centered at x) and thus $\mathbf{ifl}(\mathcal{B}) \leq 25$. As $|\mathcal{B}| = (k-2)^2$, Lemma 3.1 implies that there is a constant c such that the order of $\mathcal{B}$ is at least $k/50 - c$ and the lemma follows. ∎

Theorem 3.3. *If $\mathcal{H}$ is a hypergraph with a planar incidence graph $I(H)$, then* $\mathbf{fhw}(\mathcal{H})-1 \leq \mathbf{ghw}(\mathcal{H}) - 1 \leq \mathbf{tw}(I(\mathcal{H})) \leq 300 \cdot \mathbf{fhw}(\mathcal{H}) + c$ *for some constant $c \geq 0$.*

Proof. The left hand inequality follows directly from Lemma 2.1. Suppose now that $\mathcal{H}$ is a hypergraph where $\mathbf{fhw}(\mathcal{H}) \leq k$. By Lemmata 2.2 and 2.4, $\mathbf{fibn}(I(\mathcal{H}), V(\mathcal{H}), E(\mathcal{H})) = \mathbf{fbn}(\mathcal{H}) \leq \mathbf{fhw}(\mathcal{H}) \leq k$. By Lemmata 2.5 and 3.2, $(I(\mathcal{H}), V(\mathcal{H}), E(\mathcal{H}))$ cannot be i-contracted to an i-labeled graph (G, N, M) where G is a partially triangulated $(l \times l)$-grid, where $l = 50 \cdot k + O(1)$. By Lemma 2.3, $\mathcal{I}(\mathcal{H})$ cannot be contracted to a partially triangulated $(l \times l)$-grid and thus $I(\mathcal{H})$ excludes an $(l \times l)$-grid as a minor. From [21, (6.2)], $\mathbf{tw}(I(\mathcal{H})) \leq 6 \cdot l \leq 300 \cdot k + c$ and the result follows. ∎

Brambles in Gridoids. We call a graph G by a (k, g)-*gridoid* if it is possible to obtain a partially triangulated $(k \times k)$-grid after removing at most g edges from it (we call these edges *additional*).

Lemma 3.4. *Let (G, N, M) be an i-labeled graph where G is a (k, g)-gridoid. Then* $\mathbf{fibn}(G, N, M) \geq k/50 - c \cdot g$ *for some constant $c \geq 0$.* [Proof in [10]]

The proof of the next theorem is similar to the one of Theorem 3.3 (use Lemma 3.4 instead of Lemma 3.2 and [6, Theorem 4.12] instead of [21, (6.2)].

Theorem 3.5. *If $\mathcal{H}$ is a hypergraph with an incidence graph $I(H)$ of Euler genus at most g, then* $\mathbf{fhw}(\mathcal{H}) - 1 \leq \mathbf{ghw}(\mathcal{H}) - 1 \leq \mathbf{tw}(I(\mathcal{H})) \leq 300 \cdot g \cdot \mathbf{fhw}(\mathcal{H}) + c \cdot g$, *for some constant $c \geq 0$.*

Brambles in augmented grids. An *augmented $(r \times r)$-grid of span s* is an $r \times r$ *grid* with some extra edges such that each vertex of the resulting graph is attached to at most s non-marginal vertices of the grid.

Lemma 3.6. *If (G, N, M) is an i-labeled graph where G is an augmented $(k \times k)$-grid with span s, then* $\mathbf{fibn}(G, N, M) \geq \frac{k}{2 \cdot s^2} - c$, *for some constant $c \geq 0$.* [Proof in [10]]

As it was shown by Demaine et al. [5], every apex-minor-free graph with treewidth at least k can be contracted to a $(f(k) \times f(k))$-augmented grid of span $O(1)$ (the hidden constants in the "O"-notation depend only on the excluded apex). Because, $f(k) = \Omega(k)$ (due to the results of Demaine and Hajiaghayi in [7]), we have the following proposition.

Proposition 3.7. *Let G be an H-apex-minor-free graph of treewidth at least $c_H \cdot k$. Then G contains as a contraction an augmented $(k \times k)$-grid of span s_H, where constants c_H, s_H depend only on the size of apex graph H that is excluded.*

The proof of the next theorem is similar to the one of Theorem 3.3 (use Lemma 3.6 instead of Lemma 3.2 and Proposition 3.7 instead of [21, (6.2)].

Theorem 3.8. *If $\mathcal{H}$ is a hypergraph with an incidence graph $I(\mathcal{H})$ that is H-apex-minor-free, then* $\mathbf{fhw}(\mathcal{H}) - 1 \leq \mathbf{ghw}(\mathcal{H}) - 1 \leq \mathbf{tw}(I(\mathcal{H})) \leq c_H \cdot \mathbf{fhw}(\mathcal{H})$ *for some constant c_H that depends only on H.*

4. Hypergraphs with H-minor-free incidence graphs

The results of Theorem 3.8 cannot be extended to hypergraphs which incidence graph excludes an arbitrary fixed graph H as a minor. For example, for every integer k, it is possible to construct a hypergraph $\mathcal{H}$ with the planar incidence graph such that $\mathbf{tw}(I(\mathcal{H})) \geq k$. By adding to $\mathcal{H}$ an universal hyperedge containing all vertices of $\mathcal{H}$, we obtain a hypergraph $\mathcal{H}'$ of generalized hypertree width one. Its incidence graph $I(\mathcal{H}')$ does not contain the complete graph K_6 as a minor, however its treewidth is at least k. Despite of that, in this section we prove that if a hypergraph has H-minor-free incidence graph, then its generalized hypertree width and fractional hypertree width can be approximated by the treewidth of a graph that can be constructed from its incidence graph in polynomial time. By making use of this result we show that in this case generalized hypertree width and fractional hypertree width are up to a constant multiplicative factor from each other. Another consequence of the combinatorial result is that there is a constant factor polynomial time approximation algorithm for both parameters on this class of hypergraphs. Our proof is based on the Excluded Minor Theorem by Robertson and Seymour [23].

Graph minor theorem. Before describing the Excluded Minor Theorem we need some definitions.

Definition 4.1 (Clique-Sums)**.** Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two disjoint graphs, and $k \geq 0$ an integer. For $i = 1, 2$, let $W_i \subseteq V_i$, form a clique of size h and let G_i' be the graph obtained from G_i by removing a set of edges (possibly empty) from the clique $G_i[W_i]$. Let $F : W_1 \to W_2$ be a bijection between W_1 and W_2. We define the *h-clique-sum* of G_1 and G_2, denoted by $G_1 \oplus_{h,F} G_2$, or simply $G_1 \oplus G_2$ if there is no confusion, as the graph obtained by taking the union of G_1' and G_2' by identifying $w \in W_1$ with $F(w) \in W_2$, and by removing all the multiple edges. The image of the vertices of W_1 and W_2 in $G_1 \oplus G_2$ is called the *join* of the sum.

Note that some edges of G_1 and G_2 are not edges of G, since it is possible that these graphs had edges which were removed by clique-sum operation. Such edges are called *virtual* edges of G. We remark that $\oplus$ is not well defined; different choices of G_i' and the bijection F could give different clique-sums. A sequence of h-clique-sums, not necessarily unique, which result in a graph G, is called a *clique-sum decomposition* of G.

Definition 4.2 (h-nearly embeddable graphs)**.** Let Σ be a surface with boundary cycles $C_1, \ldots, C_h$, i.e. each cycle C_i is the border of a disc in Σ. A graph G is *h-nearly embeddable* in Σ, if G has a subset X of size at most h, called *apices*, such that there are (possibly empty) subgraphs $G_0, \ldots, G_h$ of $G - X$ such that i) $G - X = G_0 \cup \cdots \cup G_h$, ii) G_0 is embeddable in Σ, we fix an embedding of G_0, iii) graphs $G_1, \ldots, G_h$ (called *vortices*) are pairwise disjoint, iv) for $1 \leq \cdots \leq h$, let $U_i := \{u_{i_1}, \ldots, u_{i_{m_i}}\} = V(G_0) \cap V(G_i)$, G_i has a path decomposition (B_{ij}), $1 \leq j \leq m_i$, of width at most h such that a) for $1 \leq i \leq h$ and for $1 \leq j \leq m_i$ we have $u_j \in B_{ij}$, b) for $1 \leq i \leq h$, we have $V(G_0) \cap C_i = \{u_{i_1}, \ldots, u_{i_{m_i}}\}$ and the points $u_{i_1}, \ldots, u_{i_{m_i}}$ appear on C_i in this order (either if we walk clockwise or anti-clockwise).

The following proposition is known as the Excluded Minor Theorem [23] and is the cornerstone of Robertson and Seymour's Graph Minors theory.

Theorem 4.3 ([23])**.** *For every non-planar graph H, there exists an integer h, depending only on the size of H, such that every graph excluding H as a minor can be obtained by*

h-clique-sums from graphs that can be h-nearly embedded in a surface Σ in which H cannot be embedded.

Let us remark that by the result of Demaine et al. [8] such a clique-sum decomposition can be obtained in time $O(n^c)$ for some constant c which depends only from H (see also [4]).

Approximation. Let $\mathcal{H}$ be a hypergraph such that its incidence graph $G = I(\mathcal{H})$ excludes a fixed graph H as a minor. Every graph excluding a planar graph H as a minor has a constant treewidth [21]. Thus if H is planar, by Theorem 3.8, the generalized hypertree width does not exceed some constant. In what follows, we always assume that H is not planar.

By Theorem 4.3, there is an h-clique-sum decomposition of $G = G_1 \oplus G_2 \oplus \cdots \oplus G_m$ such that for every $i \in \{1, 2, \ldots, m\}$, the summand G_i can be h-nearly embedded in a surface Σ in which H can not be embedded. We assume that this clique-sum decomposition is *minimal*, in the sense that for every virtual edge $\{x, y\} \in E(G_i)$ there is an x, y-path in G with all inner vertices in $V(G) - V(G_i)$ (otherwise it is always possible to remove such edges and modify clique-sum operations correspondingly). Let A_i be the set of apices of G_i. We define $E_i = A_i \cap E(\mathcal{H})$ and $G'_i = G_i - (N_G[E_i] \cup A_i)$. For every virtual edge $\{x, y\}$ of G'_i we perform the following operation: if there is no x, y-path in $G - (N[E_i] \cup A_i)$ with all inner vertices in $G - V(G'_i)$, then $\{x, y\}$ is removed from G'_i. We denote the resulted graph by F_i.

In what remains we show that the maximal value of $\mathbf{tw}(F_i)$, where maximum is taken over all $i \in \{1, 2, \ldots, m\}$, is a constant factor approximation of generalized and fractional hypertree widths of $\mathcal{H}$. The upper bound is given by the following lemma (the proof uses results from [1]).

Lemma 4.4. $\mathbf{ghw}(\mathcal{H}) \leq 3 \cdot \max\{\mathbf{tw}(F_i) \colon i \in \{1, 2, \ldots, m\}\} + 6h + 4$. [Proof in [10]]

To prove the lower bound we need the following property of the clique-sum decomposition which was observed by Demaine and Hajiaghayi [7] (with the reference on the personal communication by Seymour).

Proposition 4.5. *Let $G = G_1 \oplus G_2 \oplus \cdots \oplus G_m$. Then every clique sum in this expression involves at most three vertices other than apices and vertices in vortices of the corresponding summand (i.e. at most three such vertices are identified by the operation).*

We also need a result roughly stating that if a graph G with a big grid as a surface minor is embedded on a surface Σ of small genus, then there is a disc in Σ containing a big part of the grid of G. This result is implicit in the work of Robertson and Seymour and there are simpler alternative proofs by Mohar and Thomassen [20, 24] (see also [6, Lemma 3.3]). We use the following variant of this result from Geelen et al. [11].

Proposition 4.6 ([11])**.** *Let g, l, r be positive integers such that $r \geq g(l + 1)$ and let G be an (r, r)-grid. If G is embedded in a surface Σ of Euler genus at most $g^2 - 1$, then some (l, l)-subgrid of G is embedded in a closed disc Δ in Σ such that the boundary cycle of the (l, l)-grid is the boundary of the disc.*

Now we are ready to prove the following lower bound.

Lemma 4.7. $\mathbf{fbn}(\mathcal{H}) \geq \varepsilon_H \cdot \max\{\mathbf{tw}(F_i) \colon i \in \{1, 2, \ldots, m\}\}$ *for some constant ε_H depending only on H.* [Proof in [10]]

Proof. Let $i \in \{1, 2, \ldots, m\}$. We assume that $G - (N[E_i] \cup A_i)$ is a connected graph which has at least one edge. (Otherwise one can consider the components of this graph separately and remove isolated vertices.) The main idea of the proof is to contract it to a planar graph with approximately the same treewidth as F_i and then apply same techniques that were used in the previous section for the planar case.

Structure of $G - (N[E_i] \cup A_i)$. Let us note that an h-clique-sum decomposition $G = G_1 \oplus G_2 \oplus \cdots \oplus G_m$ induces an h-clique-sum decomposition of $G' = G - (N[E_i] \cup A_i)$ with the summand G_i replaced by F_i. Let $G'_1, G'_2, \ldots, G'_l$ be the connected components of $G' - V(F_i)$. Every such component G'_j is attached via clique-sum to F_i by some clique Q_j of F_i. Note that cliques Q_j contain all virtual edges of F_i. We assume that each clique Q_j does not separate vertices of F_i. Otherwise, it is possible to decompose F_i into the clique-sum of graphs $F_i^{(1)} \oplus F_i^{(2)}$ with the join Q_j and prove the bound for summands and, since $\mathbf{tw}(F_i) = \max\{F_i^{(1)}, F_i^{(2)}\}$, that will prove the lemma. To simplify the structure of the graph, for every component G'_j, we contract all its edges and denote by S_j the star whose central vertex is the result of the contraction and leaves are the vertices of Q_j.

Contracting vortices. The h-nearly embedding of the graph G_i induces the h-nearly embedding of $F_i = X_0 \cup X_1 \cup \cdots \cup X_h$ without apices. Here we assume that X_0 is embedded in a surface Σ of genus depending on H and $X_1, X_2, \ldots, X_h$ are the vortices. For every vortex X_j, the vertices $V(X_0) \cap V(X_j)$ are on the boundary C_j of some face of X_0. If for a star S_k some of its leaves Q_k are in X_j or C_j, we do the following operation: if $Q_k \cap (V(X_j) - V(C_j)) \neq \emptyset$ then all edges of S_k are contracted, and if $Q_k \cap (V(X_j) - V(C_j)) = \emptyset$ but $|Q_k \cap V(C_j)| \geq 2$, then we contract all edges of S_k that are incident to the vertices of $Q_k \cap V(C_j)$. These contractions results in the contraction of some edges of F_i. Particularly, all virtual edges of X_j and C_j are contracted. Additionally, we contract all remaining edges of X_j and C_j. We perform theses contractions for all vortices of F_i and denote the result by F'_i. It follows immediately from the definition of the h-clique-sum and Proposition 4.5, that F'_i coincides with the graph obtained from F_i by contractions of all vortices X_j and boundaries of faces C_j. It can be easily seen that F'_i is embedded in Σ. It is known (see e.g. [6, 7]) that there is a positive constant a_H which depends only on H such that $\mathbf{tw}(F'_i) \geq a_H \cdot \mathbf{tw}(F_i)$.

Contracting the part that lies outside of some planar disc. Since F'_i is embedded in Σ, we have that the graph F'_i contains some $(k \times k)$-grid as a surface minor, where $k \geq b_H \cdot \mathbf{tw}(F'_i)$ for some constant b_H [6]. Combining this result with Proposition 4.6, we receive the following claim. There is a disc $\Delta \subseteq \Sigma$ such that i) the subgraph R of F'_i induced by vertices of $F'_i \cap \Delta$ is a connected graph; ii) the subgraph R' of F'_i induced by $N_{F'_i}[V(R)]$ is completely in some disc Δ'; iii) vertices of $V(R') - V(R)$ induce a cycle C which is the border of Δ', and iv) $\mathbf{tw}(R) \geq c_H \cdot \mathbf{tw}(F'_i)$ for some constant c_H. Now we treat the part of F'_i which is outside Δ exactly the same way we have treated vortices. For stars S_k intersecting $V(F'_i) - V(R')$ or C, we do the following: if $Q_k \cap (V(F'_i) - V(R')) \neq \emptyset$, then all edges of S_k are contracted, and if $Q_k \cap (V(F'_i) - V(R')) = \emptyset$ but $|Q_k \cap V(C)| \geq 2$, then all edges of S_k incident to the vertices of $Q_k \cap V(C)$ are contracted. These contractions result in the contraction of some edges of F'_i with endpoints on C or outside Δ'. Particularly, all such virtual edges are contracted. Additionally, we contract all remaining edges of $F'_i - V(R)$ and C. Thus this part of the graph is contracted to a single vertex. Denote the obtained graph X. This graph is planar, and since R is a subgraph of X, we have that $\mathbf{tw}(X) \geq \mathbf{tw}(R)$.

Embedding the stars. Some edges of X are virtual, and all such edges are in cliques Q_j. By Proposition 4.5, $|Q_j| \leq 3$. For every clique $Q = V(X) \cap Q_j$, we do the following. If $Q = \{x, y\}$, then the edge of the star S_j incident to x is contracted. If $Q = \{x, y, z\}$, then if two vertices of Q, say x and y, are joined by an edge in G, then the edge of S_j incident to z is contracted, and if there are no such edges and the triangle induced by $\{x, y, z\}$ is the boundary of some face of X, then we add a new vertex on this face, join it with x, y and z (it can be seen as S_j embedded in this face, and since our graph is i-labeled, it is assumed that this new vertex has same labels as the central vertex of S_j), and then remove virtual edges. Note that if the triangle is not a boundary of some face, then Q is a separator of our graph, but we assumed that there are no such separators. Denote by Y the obtained graph. Similar construction was used in the proof of the main theorem in [7], and by the same arguments as were used by Demaine et al. we immediately conclude that there is a positive constant d_H such that $\mathbf{tw}(X) \geq d_H \cdot \mathbf{tw}(Y)$.

Now all contractions are finished. Note that the graph Y is a planar graph which is a contraction of $G' = G - (N[E_i] \cup A_i)$. Also there is some positive constant e_H which depends only on H such that $\mathbf{tw}(Y) \geq e_H \cdot \mathbf{tw}(F_i)$. Recall that we consider the i-labeled graph $(G, V(\mathcal{H}), E(\mathcal{H}))$. By Lemma 2.4, $\mathbf{fbn}(\mathcal{H}) = \mathbf{fibn}(G, V(\mathcal{H}), E(\mathcal{H}))$. Because the sets $V(\mathcal{H})$ and $E(\mathcal{H})$ are independent, by Lemma 2.6, we have that $\mathbf{fibn}(G, V(\mathcal{H}), E(\mathcal{H})) \geq \mathbf{fibn}(G', N, M)$, where $N = V(\mathcal{H}) - (N[E_i] \cup A_i)$ and $M = E(\mathcal{H}) - (N[E_i] \cup A_i)$. By Lemma 2.5, $\mathbf{fibn}(G', N, M) \geq \mathbf{fibn}(Y, N', M')$, where N' and M' are sets which were obtained as the result of contractions of N and M. Finally, as in Theorem 3.3, one can show that $\mathbf{fibn}(Y, N', M') \geq f_H \cdot \mathbf{tw}(Y)$ for some constant f_H. By putting all these bounds together, we prove that there is a positive constant ε_H which depends only on H, such that $\mathbf{fbn}(\mathcal{H}) \geq \varepsilon_H \cdot \mathbf{tw}(F_i)$. ∎

Combining Lemmata 2.1, 2.2, 4.4, and 4.7, we obtain the following theorem.

Theorem 4.8. *$(1/c_H) \cdot w \leq \mathbf{fhw}(\mathcal{H}) \leq \mathbf{ghw}(\mathcal{H}) \leq c_H \cdot w$, where $w = \max\{\mathbf{tw}(F_i) \colon i \in \{1, 2, \ldots, m\}\}$, and c_H is a constant depending only on H.*

Remark. Notice that, by Theorem 4.8, the generalized hypertree width and the fractional hypertree width of a hypergraph with H-minor-free incidence graph may differ within a multiplicative constant factor. We stress that, as observed in [18], this is not the case for general hypergraphs.

Demaine et al. [8] (see also [4, 9, 23]) described an algorithm which constructs a clique-sum decomposition of an H-minor-free graph G on n vertices with the running time $n^{O(1)}$ (the hidden constant in the running time depends only on H). As far as we constructed summands G_i, the construction of graphs F_i can be done in polynomial time. Moreover, since the algorithm of Demaine et al. provides h-nearly embeddings of these graphs, it is possible to use it to construct a polynomial constant factor approximation algorithm for the computation of $\mathbf{tw}(F_i)$. This provides us with the main algorithmic result of this section.

Theorem 4.9. *For any fixed graph H, there is a polynomial time c_H-approximation algorithm computing the generalized hypertree width and the fractional hypertree width for hypergraphs with H-minor-free incidence graphs, where the constant c_H depends only on H.*

We finally remark that by making use of the results from [16], our results can be used not only to compute but to construct, up to constant multiplicative-factor, the corresponding decompositions.

References

[1] I. Adler, G. Gottlob, and M. Grohe, *Hypertree width and related hypergraph invariants*, European J. Combin., 28 (2007), pp. 2167–2181.

[2] A. K. Chandra and P. M. Merlin, *Optimal implementation of conjunctive queries in relational data bases*, in STOC'77, ACM, 1977, pp. 77–90.

[3] C. Chekuri and A. Rajaraman, *Conjunctive query containment revisited*, Theoret. Comput. Sci., 239 (2000), pp. 211–229.

[4] A. Dawar, M. Grohe, and S. Kreutzer, *Locally excluding a minor*, in LICS'07, IEEE Computer Society, 2007, pp. 270–279.

[5] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos, *Bidimensional parameters and local treewidth*, SIAM J. Discrete Math., 18 (2004/05), pp. 501–511.

[6] ———, *Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs*, J. ACM, 52 (2005), pp. 866–893.

[7] E. D. Demaine and M. Hajiaghayi, *Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality*, in SODA'05, ACM, 2005, pp. 682–689.

[8] E. D. Demaine, M. T. Hajiaghayi, and K. ichi Kawarabayashi, *Algorithmic graph minor theory: Decomposition, approximation, and coloring*, in FOCS'05, IEEE Computer Society, 2005, pp. 637–646.

[9] U. Feige, M. Hajiaghayi, and J. R. Lee, *Improved approximation algorithms for minimum weight vertex separators*, SIAM J. Computing, 38 (2008), pp. 629–657.

[10] F. V. Fomin, P. A. Golovach, and D. M. Thilikos, *Approximating acyclicity parameters of sparse hypergraphs*, CoRR, abs/0809.3646 (2008).

[11] J. F. Geelen, R. B. Richter, and G. Salazar, *Embedding grids in surfaces*, European J. Combin., 25 (2004), pp. 785–792.

[12] G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello, *Hypertree decompositions: Structure, algorithms, and applications*, in WG'05, vol. 3787 of Lecture Notes in Computer Science, Springer, 2005, pp. 1–15.

[13] G. Gottlob, N. Leone, and F. Scarcello, *A comparison of structural CSP decomposition methods*, Artificial Intelligence, 124 (2000), pp. 243–282.

[14] ———, *The complexity of acyclic conjunctive queries*, J. ACM, 48 (2001), pp. 431–498.

[15] ———, *Hypertree decompositions and tractable queries*, J. Comput. System Sci., 64 (2002), pp. 579–627.

[16] ———, *Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width*, J. Comput. System Sci., 66 (2003), pp. 775–808.

[17] G. Gottlob, Z. Miklós, and T. Schwentick, *Generalized hypertree decompositions: NP-hardness and tractable variants*, in PODS'07, ACM, 2007, pp. 13–22.

[18] M. Grohe and D. Marx, *Constraint solving via fractional edge covers*, in SODA'06, ACM, 2006, pp. 289–298.

[19] P. G. Kolaitis and M. Y. Vardi, *Conjunctive-query containment and constraint satisfaction*, J. Comput. System Sci., 61 (2000), pp. 302–332.

[20] B. Mohar, *Combinatorial local planarity and the width of graph embeddings*, Canad. J. Math., 44 (1992), pp. 1272–1288.

[21] N. Robertson, P. Seymour, and R. Thomas, *Quickly excluding a planar graph*, J. Combin. Theory Ser. B, 62 (1994), pp. 323–348.

[22] N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, J. Algorithms, 7 (1986), pp. 309–322.

[23] ———, *Graph minors. XVI. Excluding a non-planar graph*, J. Combin. Theory Ser. B, 89 (2003), pp. 43–76.

[24] C. Thomassen, *A simpler proof of the excluded minor theorem for higher surfaces*, J. Combin. Theory Ser. B, 70 (1997), pp. 306–311.

[25] M. Yannakakis, *Algorithms for acyclic database schemes*, in VLDB'81, IEEE Computer Society, 1981, pp. 82–94.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 457–468
www.stacs-conf.org

OPTIMAL CACHE-AWARE SUFFIX SELECTION

GIANNI FRANCESCHINI [1] AND ROBERTO GROSSI [2] AND S. MUTHUKRISHNAN [3]

[1] Dipartimento di Informatica, Università di Roma "La Sapienza", and PINT, Primorska University
E-mail address: francesc@di.uniroma1.it

[2] Dipartimento di Informatica, Università di Pisa
E-mail address: grossi@di.unipi.it

[3] Google Inc., NY
E-mail address: muthu@google.com

Abstract. Given string $S[1..N]$ and integer k, the *suffix selection* problem is to determine the kth lexicographically smallest amongst the suffixes $S[i \dots N]$, $1 \leq i \leq N$. We study the suffix selection problem in the cache-aware model that captures two-level memory inherent in computing systems, for a *cache* of limited size M and block size B. The complexity of interest is the number of block transfers. We present an optimal suffix selection algorithm in the cache-aware model, requiring $\Theta(N/B)$ block transfers, for any string S over an unbounded alphabet (where characters can only be compared), under the common tall-cache assumption (i.e. $M = \Omega\left(B^{1+\epsilon}\right)$, where $\epsilon < 1$). Our algorithm beats the bottleneck bound for permuting an input array to the desired output array, which holds for nearly any nontrivial problem in hierarchical memory models.

1. Introduction

Background: Selection vs Sorting. A collection of N *numbers* can be sorted using $\Theta(N \log N)$ comparisons. On the other hand, the famous five-author result [2] from early 70's shows that the problem of *selection* — choosing the kth smallest number — can be solved using $O(N)$ comparisons in the worst case. Thus, selection is provably simpler than sorting in the comparison model.

Consider a sorting vs selection question for strings. Say $S = S[1 \cdots N]$ is a string. The *suffix sorting* problem is to sort the suffixes $S[i \cdots N]$, $i = 1, \dots, N$, in the *lexicographic* order. In the comparison model, we count the number of character comparisons. Suffix sorting can be performed with $O(N \log N)$ comparisons using a combination of character sorting and classical data structure of suffix arrays or trees [11, 9, 4]. There is a lower bound of $\Omega(N \log N)$ since sorting suffixes ends up sorting the characters. For the related *suffix selection* problem where the goal is to output the kth lexicographically smallest suffix of S, the result in [6] recently gave an optimal $O(N)$ comparison-based algorithm, thereby showing that suffix selection is provably simpler than suffix sorting.

The first two authors have been partially supported by the MIUR project MAINSTREAM.

The Model. Time-tested architectural approaches to computing systems provide two (or more) levels of memory: the highest one with a limited amount of fast memory; the lowest one with slow but large memory. The CPU can only access input stored on the fastest level. Thus, there is a continuous exchange of data between the levels. For cost and performance reasons, data is exchanged in fixed-size blocks of contiguous locations. These transfers may be triggered automatically like in internal CPU caches, or explicitly, like in the case of disks; in either case, more than the number of computing operations executed, the number of block transfers required is the actual bottleneck.

Formally, we consider the model that has two memory levels. The *cache* level contains M locations divided into blocks (or cache lines) of B contiguous locations, and the *main memory* level can be arbitrarily large and is also divided into blocks. The processing unit can address the locations of the main memory but it can process only the data residing in cache. The algorithms that know and exploit the two parameter M and B, and optimize the number of block transfers are *cache-aware*. This model includes the classical External Memory model [1] as well as the well-known *Ideal-Cache model* [7].

Motivation. Suffix selection as a problem is useful in analyzing the order statistics of suffixes in a string such as the extremes, medians and outliers, with potential applications in bioinformatics and information retrieval. A quick method for finding say the suffixes of rank $i(n/10)$ for each integer i, $0 \leq i \leq 10$, may be used to partition the space of suffixes for understanding the string better, load balancing and parallelization. But in these applications, such as in bioinformatics, the strings are truly massive and unlikely to fit in the fastest levels of memory. Therefore it is natural to analyze them in a hierarchical memory model.

Our primary motivation however is really theoretical. Since the inception of the first block-based hierarchical memory model ([1],[10]), it has been difficult to obtain "golden standard" algorithms i.e., those using just $O(N/B)$ block transfers. Even the simplest permutation problem (PERM henceforth) where the output is a specified permutation of the input array, does not have such an algorithm. In the standard RAM model, PERM can be solved in $O(N)$ time. In both the Ideal-Cache and External Memory models, the complexity of this problem is denoted $\text{PERM}(N) = \Theta\left(\min\left\{N, (N/B\log_{M/B} N/B\right\}\right)$. Nearly any nontrivial problem one can imagine from list ranking to graph problems such as Euler tours, DFS, connected components etc., sorting and geometric problems have the lower bound of $\text{PERM}(N)$, even if they take $O(N)$ time in the RAM model, and therefore do not meet the "golden standard". Thus the lower bound for PERM is a terrible bottleneck for block-based hierarchical memory models.

The outstanding question is, much as in the comparison model, is suffix selection provably simpler than suffix sorting in the block-based hierarchical memory models? Suffix sorting takes $\Theta\left((N/B)\log_{M/B}(N/B)\right)$ block transfers [5]. Proving any problem to be simpler than suffix sorting therefore requires one to essentially overcome the PERM bottleneck.

Our Contribution. We present a suffix selection algorithm with optimal cache complexity. Our algorithm requires $\Theta(N/B)$ block transfers, for any string S over an unbounded alphabet (where characters can only be compared) and under the common tall-cache assumption, that is $M = \Omega\left(B^{1+\epsilon}\right)$ with $\epsilon < 1$. Hence, we meet the "golden standard"; we

beat the PERM bottleneck and consequently, prove that suffix selection is easier than suffix sorting in block-based hierarchical memory models.

Overview. Our high level strategy for achieving an optimal cache-aware suffix selection algorithm consists of two main objectives.

In the first objective, we want to efficiently reduce the number of candidate suffixes from N to $O(N/B)$, where we maintain the invariant that the wanted kth smallest suffix is surely one of the candidate suffixes.

In the second objective, we want to achieve a cache optimal solution for the *sparse suffix selection problem*, where we are given a subset of $O(N/B)$ suffixes including also the wanted kth suffix. To achieve this objective we first find a simpler approach to suffix selection for the standard comparison model. (The only known linear time suffix selection algorithm for the comparison model [6] hinges on well-known algorithmic and data structural primitives whose solutions are inherently cache inefficient.) Then, we modify the simpler comparison-based suffix selection algorithm to exploit, in a cache-efficient way, the hypothesis that $O(N/B)$ (known) suffixes are the only plausible candidates.

Map of the paper. We will start by describing the new simple comparison-based suffix selection algorithm in Section 2. This section is meant to be intuitive. We will use it to derive a cache-aware algorithm for the sparse suffix selection problem in Section 3. We will present our optimal cache-aware algorithm for the general suffix selection problem in Section 4.

2. A Simple(r) Linear-Time Suffix Selection Algorithm

We now describe a simple algorithm for selecting the kth lexicographically smallest suffix of S in main memory. We give some intuitions on the central notion of `work`, and some definitions and notations used in the algorithm. Next, we show how to perform main iteration, called *phase transition*. Finally, we present the invariants that are maintained in each phase transition, and discuss the correctness and the complexity of our algorithm.

Notation and intuition. Consider the regular linear-time selection algorithm [2], hereafter called BFPRT. Our algorithm for a string $S = S[1 \dots N]$ uses BFPRT as a black box.[1] Each run of BFPRT permits to discover a longer and longer prefix of the (unknown) kth lexicographically smallest suffix of S. We need to carefully orchestrate the several runs of BFPRT to obtain a total cost of $O(N)$ time. We use $S =$ `bbbabbbbbaa$`, where $n = 12$, as an illustrative example, and show how to find the median suffix (hence, $k = n/2 = 6$).

Phases and phase transitions. We organize our computation so that it goes through *phases*, numbered $t = 0, 1, 2, \dots$ and so on. In phase t, we know that a certain string, denoted σ_t, is a prefix of the (unknown) kth lexicographically smallest suffix of S. Phase $t = 0$ is the initial one: we just have the input string S and no knowledge, i.e., σ_0 is the empty string. For $t \geq 1$, a main iteration of our algorithm goes from phase $t-1$ to phase t and is termed *phase transition* $(t-1 \rightarrow t)$: it is built around the tth run of BFPRT on a suitable subset of the suffixes of S. Note that $t \leq N$, since we ensure that the condition $|\sigma_{t-1}| < |\sigma_t|$ holds, namely, each phase transition extends the known prefix by at least one symbol.

[1] In the following, we will assume that the last symbol in S is an endmarker $S[N] = \$$, smaller than any other symbol in S.

Phase transition $(0 \to 1)$. We start out with phase 0, where we run BFPRT on the individual symbols of S, and find the symbol α of rank k in S (seen as a multiset). Hence we know that $\sigma_1 = \alpha$, and this fact has some implications on the set of suffixes of S. Let s_i denote the ith suffix $S[i \dots N]$ of S, for $1 \le i \le N$, and w_i be a special prefix of s_i called `work`. We anticipate that the `work`s play a fundamental role in attaining $O(N)$ time. To complete the phase transition, we set $w_i = S[i]$ for $1 \le i \le N$, and we call *degenerate* the `work`s w_i such that $w_i \neq \alpha$. (Note that degenerate `work`s are only created in this phase transition.) We then partition the suffixes of S into two disjoint sets:

- The set of *active suffixes*, denoted by $\mathcal{A}_1$—they are those suffixes s_i such that $w_i = \sigma_1 = \alpha$.
- The set of *inactive suffixes*, denoted by $\mathcal{I}_1$ and containing the rest of the suffixes—none of them is surely the kth lexicographically smallest suffix in S.

In our example ($k = 6$), we have $\sigma_1 = \alpha = \texttt{b}$ and, for $i = 1, 2, 3, 5, 6, 7, 8, 9$, $w_i = \texttt{b}$ and $s_i \in \mathcal{A}_1$. Also, we have $s_j \in \mathcal{I}_1$ for $j = 4, 10, 11, 12$, where $w_4 = w_{10} = w_{11} = \texttt{a}$ and $w_{12} = \$$ are degenerate `work`s.

A comment is in order at this point. We can compare any two `work`s in constant time, where the outcome of the comparison is ternary $[<, =, >]$. While this observation is straightforward for this phase transition, we will be able to extend it to longer `work`s in the subsequent transitions. Let us discuss the transition from phase 1 to phase 2 to introduce the reader to the main point of the algorithm.

Phase transition $(1 \to 2)$. If $|\mathcal{A}_1| = 1$, we are done since there is only one active suffix and this should be the kth smallest suffix in S. Otherwise, we exploit the total order on the current `work`s. Letting z_1 be the number of `work`s smaller than the current prefix σ_1, our goal becomes how to find the $(k - z_1)$th smallest suffix in $\mathcal{A}_1$. In particular, we want a longer prefix σ_2 and the new set $\mathcal{A}_2 \subseteq \mathcal{A}_1$.

To this end, we need to extend some of the `work`s of the active suffixes in $\mathcal{A}_1$. Consider a suffix $s_i \in \mathcal{A}_1$. In order to extend its `work` w_i, we introduce its *prospective* `work`. Recall that $w_i = \sigma_1 = \alpha = S[i]$. If $w_{i+1} = S[i+1] \neq \alpha$ (hence, s_{i+1} is inactive in our terminology), the prospective `work` for s_i is the concatenation $w_i w_{i+1}$, where $s_{i+1} \in \mathcal{I}_1$. Otherwise, since $w_i = w_{i+1}$ (and so $s_{i+1} \in \mathcal{A}_1$), we consider $i + 2, i + 3$, and so on, until we find the first $i + r$ such that $w_i \neq w_{i+r}$ (and so $s_{i+r} \in \mathcal{I}_1$). In the latter case, the prospective `work` for s_i is the concatenation $w_i w_{i+1} \cdots w_{i+r}$, where $w_i = w_{i+1} = \cdots = w_{i+r-1} = \sigma_1 = \alpha$ and their corresponding suffixes are active, while $w_{i+r} \neq \sigma_1$ is different and corresponds to an inactive suffix.

In any case, each prospective `work` is a sequence of `work`s of the form $\alpha^r \beta = \sigma_1^r \beta$, where $r \ge 1$ and $\beta \neq \alpha$. The reader should convince herself that any two prospective `work`s can be compared in $O(1)$ time. We exploit this fact by running BFPRT on the set $\mathcal{A}_1$ of active suffixes and, whenever BFPRT requires to compare any two $s_i, s_j \in \mathcal{A}_1$, we compare their prospective `work`s. Running time is therefore $O(|\mathcal{A}_1|)$ if we note that prospective `work`s can be easily identified by a scan of $\mathcal{A}_1$: if $w_i w_{i+1} \cdots w_{i+r}$ is the prospective `work` for s_i, then $w_{i+1} \cdots w_{i+r}$ is the prospective `work` for s_{i+1}, and so on. In other words, a consecutive run of prospective `work`s forms a *collision*, which is informally a maximal concatenated sequence of `work`s equal to σ_1 terminated by a `work` different from σ_1 (this notion will be described formally in Section 2).

After BFPRT completes its execution, we know the prospective `work` that is a prefix of the (unknown) kth suffix in S. That prospective `work` becomes σ_2 and $\mathcal{A}_2$ is made up of the the suffixes in $\mathcal{A}_1$ such that their prospective `work` equals σ_2 (and we also set z_2).

In our example, $z_1 = 3$, and so we look for the third smaller suffix in $\mathcal{A}_1$. We have the following prospective `work`s: one collision is made up of $p_1 =$ `bbba`, $p_2 =$ `bba`, and $p_3 =$ `ba`; another collision is made up of $p_5 =$ `bbbbba`, $p_6 =$ `bbbba`, $p_7 =$ `bbba`, $p_8 =$ `bba`, and $p_9 =$ `ba`. Algorithm BFPRT discovers that `bba` is the third prospective `work` among them, and so $\sigma_2 =$ `bba` and $\mathcal{A}_2 = \{s_2, s_8\}$ (and $z_2 = 5$).

How to maintain the `work`*s.* Now comes the key point in our algorithm. For each suffix $s_i \in \mathcal{A}_2$, we update its `work` to be $w_i = \sigma_2$ (whereas it was $w_i = \sigma_1$ in the previous phase transition, so it is now longer). For each suffix $s_i \in \mathcal{A}_1 - \mathcal{A}_2$, instead, we leave its `work` w_i *unchanged*. Note this is the key point: although s_i can share a longer prefix with σ_2, the algorithm BFPRT has indirectly established that s_i cannot have σ_2 as a prefix, and we just need to record a Boolean value for w_i, indicating if w_i is either lexicographically smaller or larger than σ_2. We can stick to w_i unchanged, and discard its prospective `work`, since s_i becomes inactive and is added to $\mathcal{I}_2$. In our example, $w_2 = w_8 =$ `bba`, while the other `work`s are unchanged (i.e, $w_3 =$ `b` while $p_3 =$ `ba`, $w_5 =$ `b` while $p_5 =$ `bbbbba`, and so on).

In this way, we can maintain a *total order on the* `work`*s*. If two `work`s are of equal length, we declare that they are equal according to the symbol comparisons that we have performed so far, unless they are degenerate—in the latter case they can be easily compared as single symbols. If two `work`s are of different length, say $|w_i| < |w_j|$, then s_i has been discarded by BFPRT in favor of s_j in a certain phase, so we surely know which one is smaller or larger. In other words, when we declare two `work`s to be equal, we have not yet gathered enough symbol comparisons to distinguish among their corresponding suffixes. Otherwise, we have been able to implicitly distinguish among their corresponding suffixes. In our example, $w_3 < w_2$ because they are of different length and BFPRT has established this disequality, while we declare that $w_3 = w_5$ since they have the same length. Recall that the total order on the `work`s is needed for comparing any two prospective `work`s in $O(1)$ time as we proceed in the phase transitions. The `work`s exhibit some other strong properties that we point out in the invariants described in Section 2.

Time complexity. From the above discussion, we spend $O(|\mathcal{A}_1)|)$ time for phase transition $(1 \to 2)$. We present a charging scheme to pay for that. `work`s come again into play for an amortized cost analysis. Suppose that, in phase 0, we initially assign each suffix s_i two kinds of credits to be charged as follows: $O(1)$ credits of the first kind when s_i becomes inactive, and further $O(1)$ credits of the second kind when s_i is already inactive but its `work` w_i becomes the terminator of the prospective `work` of an active suffix. Note that w_i is incapsulated by the prospective `work` of that suffix (which survives and becomes part of $\mathcal{A}_2$).

Now, when executing BFPRT on $\mathcal{A}_1$ as mentioned above, we have that at most one prospective `work` *survives* in each collision and the corresponding suffix becomes part of $\mathcal{A}_2$. We therefore charge the cost $O(|\mathcal{A}_1|)$ as follows. We take $\Theta(|\mathcal{A}_1| - |\mathcal{A}_2|)$ credits of the first kind from the $|\mathcal{A}_1| - |\mathcal{A}_2| \geq 0$ active suffixes that become inactive at the end of the phase transition. We also take $\Theta(|\mathcal{A}_2|)$ credits from the $|\mathcal{A}_2|$ inactive suffixes whose `work` terminates the prospective `work` of the survivors. In our example, the $\Theta(|\mathcal{A}_1| - |\mathcal{A}_2|)$ credits are taken from s_1, s_3, s_5, s_6, s_7, and s_9, while $\Theta(|\mathcal{A}_2|)$ credits are taken from s_4 and s_{10}.

At this point, it should be clear that, in our example, the next phase transition ($2 \to 3$) looks for the $(k - z_2)$th smaller suffix in $\mathcal{A}_2$ by executing BFPRT in $O(|\mathcal{A}_2|)$ time on the prospective `work`s built with the runs of consecutive occurrences of the `work` $\sigma_2 =$ `bba` into S. We thus identify `bbaa`\$ (with $\sigma_3 =$ `bbaa`) as the median suffix in S.

Phase transition ($t-1 \to t$) for $t \geq 1$. We are now ready to describe the generic phase transition ($t-1 \to t$) more formally in terms of the active suffixes in $\mathcal{A}_{t-1}$ and the inactive ones in $\mathcal{I}_{t-1}$, where $t \geq 1$.

The input for the phase transition is the following: (a) the current prefix σ_{t-1} of the (unknown) kth lexicographically smallest suffix in S; (b) the set $\mathcal{A}_{t-1}$ of currently active suffixes; (c) the number z_{t-1} of suffixes in $\mathcal{I}_{t-1}$ whose `work` is smaller than that of the suffixes in $\mathcal{A}_{t-1}$ (hence, we have to find the $(k - z_{t-1})$th smallest suffix in $\mathcal{A}_{t-1}$); and (d) a Boolean vector whose ith element is false (resp., true) iff, for suffix $s_i \in \mathcal{I}_{t-1}$, the algorithm BFPRT has determined that its `work` w_i is smaller (resp., larger) than σ_{t-1}. The output of the phase transition are data (a)–(d) above, updated for phase t.

We now define collisions and prospective `work`s in a formal way. We say that two suffixes $s_i, s_j \in \mathcal{A}_t$ *collide* if their `work`s w_i and w_j are adjacent as substrings in S, namely, $|i - j| = |w_i| = |w_j|$. A *collision* C is the maximal subsequence $w_{l_1} w_{l_2} \cdots w_{l_r}$, such that $w_{l_1} = w_{l_2} = \cdots = w_{l_r} = \sigma_t$, where the active suffixes s_{l_f} and $s_{l_{f+1}}$ collide for any $1 \leq f < r$. For our algorithm, a *collision* can also be a degenerate sequence of just one active suffix s_i (since its `work` does not collide with that of any other active suffix).

The *prospective* `work` of a suffix $s_i \in \mathcal{A}_{t-1}$, denoted by p_i, is defined as follows. Consider the collision C to which s_i belongs. Suppose that s_i is the hth active suffix (from the left) in C, that is, $C = w_{l_1} w_{l_2} \cdots w_{l_{h-1}} w_i w_{l_{h+1}} \cdots w_{l_{r-1}} w_{l_r}$. Consider the suffix $s_u \in \mathcal{I}_{t-1}$ adjacent to w_{l_r} (because of the definition of collision, s_u must be an inactive suffix following w_{l_r}). We define the prospective `work` of s_i, to be the string $p_i = w_i w_{l_{h+1}} \cdots w_{l_{r-1}} w_{l_r} w_u$. Note that $w_i = w_{l_{h+1}} = \cdots = w_{l_{r-1}} = w_{l_r} = \sigma_{t-1}$ since their corresponding suffixes are all active, while w_u is shorter. In other words, $p_i = \sigma_{t-1}^{r-h} w_u$, with $|w_u| < |\sigma_{t-1}|$.

Lemma 2.1. *For any two suffixes $s_i, s_j \in \mathcal{A}_t$, we can compare their prospective* `work`s *p_i and p_j in $O(1)$ time.*

We now give the steps for the phase transition. Note that we can maintain $\mathcal{A}_{t-1}$ in monotone order of suffix position (i.e., $i < j$ implies that s_i comes first than s_j in $\mathcal{A}_{t-1}$).

(1) Scan the active set $\mathcal{A}_{t-1}$ and identify its collisions and the set $\mathscr{T}$ containing all the suffixes $s_u \in \mathcal{I}_{t-1}$ such that w_u immediately follows a collision. For any suffix s_i in $\mathcal{A}_{t-1}$, determine its prospective `work` p_i using the collisions and $\mathscr{T}$.

(2) Apply algorithm BFPRT to the set $\{p_i\}_{s_i \in \mathcal{A}_{t-1}}$ using the constant-time comparison as stated in Lemma 2.1. In this way, find the $(k - z_{t-1})$th lexicographically smallest prospective `work` p, and the corresponding set $\mathcal{A}_t = \{s_i \in \mathcal{A}_{t-1} \mid p_i = p\}$ of active suffixes whose prospective `work`s match p.
   (a) If $|\mathcal{A}_t| = 1$, stop the computation and return the singleton $s_i \in \mathcal{A}_t$ as the kth smallest suffix in S.
   (b) If $|\mathcal{A}_t| > 1$, set $\sigma_t = p$ (and update z_t accordingly).

(3) For each $s_i \in \mathcal{A}_t$: Let $p = w_i w_{l_{h+1}} \cdots w_{l_r} w_u$ be its prospective `work`, where $s_u \in \mathscr{T}$. Set its new `work` to be $w_i = p = \sigma_t$.

(4) For each $s_j \in \mathcal{A}_{t-1} - \mathcal{A}_t$, leave its `work` w_j *unchanged* and, as a byproduct of running BFPRT in step 2, update position j of the Boolean vector (d) given in input, so as to record the fact that w_j is lexicographically smaller or larger than σ_t.

Lemma 2.2. *Executing phase transition* $(t-1 \to t)$ *with* $t \geq 1$*, requires* $O(|\mathcal{A}_{t-1}|)$ *time in the worst case.*

Invariants for phase *t*. Before proving the correctness and the complexity of our algorithm, we need to establish some invariants that are maintained through the phase transitions. We say that w_i is *maximal* if there does not exist another suffix s_j such that w_j contains w_i, namely, such that $j < i$ and $i + |w_i| \leq j + |w_j|$. For any $t \geq 1$, the following invariants holds (where $\mathcal{A}_0$ is trivially the set of all the suffixes):

(i) [prefixes]: σ_{t-1} and σ_t are prefixes of the (unknown) kth smallest suffix of S, and $|\sigma_{t-1}| < |\sigma_t|$.

(ii) [works]: For any suffix s_i, its `work` w_i is either degenerate (a single mismatching symbol) or $w_i = \sigma_{t'}$ for a phase $t' \leq t$. Moreover, $w_i = \sigma_t$ iff $s_i \in \mathcal{A}_t$.

(iii) [comparing]: For any s_i and s_j, $|w_i| \neq |w_j|$ implies that we know whether $w_i < w_j$ or $w_i > w_j$.

(iv) [nesting]: For any two suffixes s_i and s_j, their `works` w_i and w_j do not overlap (either they are disjoint or one is contained within the other). Namely, $i > j$ implies $i + |w_i| \leq j + |w_j|$ or $i \geq j + |w_j|$.

(v) [covering]: The `works` of the active suffixes are all maximal and, together with the maximal `works` generated by the inactive suffixes, form a *non-overlapping covering* of S (i.e. $S = w_{i_1} w_{i_2} \cdots w_{i_r}$, where $i_1 < i_2 < \cdots < i_r$ and either $s_{i_j} \in \mathcal{A}_t$, or $s_{i_j} \in \mathcal{I}_t$ and w_{i_j} is maximal, for $1 \leq j \leq r$).

Lemma 2.3. *After phase transition* $(t-1 \to t)$ *with* $t \geq 1$*, the invariants (i)–(v) are maintained.*

Theorem 2.4. *The algorithm terminates in a phase* $t \leq N$*, and returns the* k*th lexicographically smallest suffix.*

Theorem 2.5. *Our suffix selection algorithm requires* $O(N)$ *time in the worst case.*

This simpler suffix selection algorithm is still cache "unfriendly". For example, it requires $O(N)$ block transfers with a string S with period length $\Theta(B)$ (if S is a prefix of g^i for some integer i, then g is a period of S).

3. Cache-Aware Sparse Suffix Selection

In *the sparse suffix selection problem*, along with the string S and the rank k of the suffix to retrieve, we are also given a set $\mathscr{K}$ of suffixes such that $|\mathscr{K}| = O(N/B)$ and the kth smallest suffix belongs in $\mathscr{K}$. We want to find the wanted suffix in $O(N/B)$ block transfers using the ideas of the algorithm described in Section 2.

Consider first a particular situation in which the suffixes are equally spaced B positions each other. We can split S into blocks of size B, so that S is conceptually a string of N/B metacharacters and each suffix starts with a metacharacters. This is a fortunate situation since we can apply the algorithm described in Section 2 as is, and solve the problem in the claimed bound. The nontrivial case is when the suffixes can be in arbitrary positions.

Hence, we revisit the algorithm described in Section 2 to make it more cache efficient. Instead of trying to extend the `work` of an active suffix s_i by just using the `works` of the following inactive suffixes, we try to batch these `works` in a sufficiently long segment, called *reach*. Intuitively, in a step similar to step 2 of the algorithm in Section 2, we could first apply the BFPRT algorithm to the set of reaches. Then, after we select a subset of equal reaches, and the corresponding subset of active suffixes, we could extend their `works` using

their reaches. This could cause collisions between the suffixes and they could be managed in a way similar to what we did in Section 2. This yields the notion of super-phase transition.

Super-phase transition. The purpose of a super-phase is to group consecutive phases together, so that we maintain the same invariants as those defined in Section 2. However, we need further concepts to describe the transition between super-phases. We number the super-phases according to the numbering of phases. We call a super-phase m if the *first* phase in it is m (in the overall numbering of phases).

Reaches, pseudo-collisions and prospective reaches. Consider a generic super-phase m. Recall that, by the invariant (v) in Section 2, the phase transitions maintain the string S partitioned into maximal `work`s. We need to define a way to access enough (but not too many) consecutive "lookahead" `work`s following each active suffix, before running the super-phase. Since some of these active suffixes will become inactive during the phases that form the super-phase, we cannot prefetch too many such `work`s (and we cannot predict which ones will be effectively needed). This idea of prefetching leads to the following notion.

For any active suffix $s_i \in \mathcal{A}_m$, the *reach* of s_i, denoted by r_i, is the *maximal* sequence of consecutive `work`s $w_{l_1} w_{l_2} \cdots w_{l_f}$ such that

(i) $i < l_1 < l_2 < \cdots < l_f$ and $l_f - l_1 < B$;
(ii) w_i and w_{l_1} are adjacent and, for $1 < x \leq f$, $w_{l_{x-1}}$ and w_{l_x} are adjacent in S;
(iii) if s_j is the leftmost active suffix in $S[i+1 \ldots N]$, then $l_f \leq j$.

We call a reach *full* if $l_f < j$ in condition (iii), namely, we do not meet an active suffix while loading the reach. Since we know how to compare two `work`s, we also know how to compare any two reaches r_i, r_j, seen as sequences of `work`s. We have the following.

Lemma 3.1. *For any two reaches r_i and r_j, such that $|r_i| < |r_j|$, we have that r_i cannot be a prefix of r_j.*

Using reaches, we must possibly handle the collisions that may occur in an arbitrary phase that is internal to the current super-phase. We therefore introduce a notion of collision for reaches that is called pseudo-collision because it does not necessarily implies a collision.

For any two reaches r_i, r_j such that $i < j$, we say that r_i and r_j *pseudo-collide* if $r_i = r_j$ and the last `work` of r_i *is* w_j itself (not just *equal to* w_j). Thus, the last `work` of r_j is active and equal to w_i and w_j. Certainly, the fact that r_i and r_j pseudo-collide during a super-phase does not necessarily imply that the `work`s w_i and w_j collide in one of its phases. A *pseudo-collision* $PC(l)$ is a maximal sequence $r_{l_1} r_{l_2} \cdots r_{l_a}$ such that r_{l_f} and $r_{l_{f+1}}$ pseudo-collide, for any $1 \leq f < a$. For our algorithm, a degenerate pseudo-collision is a sequence of just one reach.

Let us consider an active suffix s_i and the pseudo-collision to which r_i belongs. Let us suppose that the pseudo-collision is $r_{l_1} r_{l_2} \cdots r_{l_{f-1}} r_i r_{l_{f+1}} \cdots r_{l_a}$ (i.e. r_i is the fth reach). Also, let us consider the reach r_u of the last `work` w_u that appears in r_{l_a} (by the definition of pseudo-collision, we know that the last `work` w_u of r_{l_a} is equal to its first `work`, so s_u is active and has a reach). The *prospective reach* of an active `work` w_i, denoted by pr_i, is the sequence $r_i r_{l_{f+1}} \cdots r_{l_a} tail(pr_i)$, where $tail(pr_i) = lcp(r_i, r_u)$ is the *tail* of pr_i and denotes the longest initial sequence of `work`s that is common to both r_i and r_u. Analogously to prospective `work`s, we can define a total order on the prospective reaches. The *multiplicity* of pr_i, denoted by $mult(pr_i)$, is $a - f + 1$ (that is the number of reaches following r_i in the pseudo-collision plus r_i).

Lemma 3.2. *If the invariants for the phases hold for the current super-phase then, for any two reaches r_i and r_j such that $r_i = r_j$, we have that their prospective reaches pr_i and pr_j can be compared in $O(1)$ time, provided we know the lengths of tail (pr_i) and tail (pr_j).*

Super-phase transition $(m \to m')$. The transition from a super-phase m to the next super-phase m' *emulates* what happens with phases $m, m+1, \ldots, m'$ in the algorithm of Section 2, but using $O(N/B)$ block transfers.

(1) For each active suffix s_i, we create a pointer to its reach r_i.
(2) We find the $(k - z_m)$th lexicographically smallest reach ρ using BFPRT on the $O(N/B)$ pointers to reaches created in the previous step. The sets $\mathscr{R}_= = \{s_i \mid s_i \text{ is active and } r_i = \rho\}$, $\mathscr{R}_< = \{s_i \mid s_i \text{ is active and } r_i < \rho\}$, and $\mathscr{R}_> = \{s_i \mid s_i \text{ is active and } r_i > \rho\}$ are thus identified, and, for any $s_i \in \mathscr{R}_< \cup \mathscr{R}_>$, the length of $lcp(r_i, \rho)$.[2] If $|\mathscr{R}_=| = 1$, we stop and return s_i, such that $s_i \in \mathscr{R}_=$, as the kth smallest suffix in S.
(3) For any $s_i \in \mathscr{R}_=$, we compute its prospective reach pr_i.
(4) We find the $(k - z_m - |\mathscr{R}_<|)$th lexicographically smallest prospective reach π among the ones in $\{pr_i \mid s_i \in \mathscr{R}_=\}$, thus obtaining $\mathscr{P}_= = \{s_i \mid s_i \text{ is active and } pr_i = \pi\}$, $\mathscr{P}_< = \{s_i \mid s_i \text{ is active and } pr_i < \pi\}$, $\mathscr{P}_> = \{s_i \mid s_i \text{ is active and } pr_i > \pi\}$, and, for any $s_i \in \mathscr{P}_< \cup \mathscr{P}_>$, the length of $lcp(pr_i, \pi)$. If $|\mathscr{P}_=| = 1$, we stop and return s_i, such that $s_i \in \mathscr{P}_=$, as the kth smallest suffix in S.

Theorem 3.3. *The sparse suffix selection problem can be solved using $O(N/B)$ block transfers in the worst case.*

4. Optimal Cache-Aware Suffix Selection

The approach in Sec. 3 does not work if the number of input active suffixes is $\omega(N/B)$. The process would cost $O\left(\frac{N}{B} \log B\right)$ block transfers (since it would take $\Omega(\log B)$ transitions to finally have $O(N/B)$ active suffixes left). However, if we were able to find a set $\mathscr{K}$ of $O(N/B)$ suffixes such that one of them is the kth smallest, we could solve the problem with $O(N/B)$ block transfers using the algorithm in Sec. 3. In this section we show how to compute such a set $\mathscr{K}$.

Basically, we consider all the substrings of length B of S and we select a suitable set of $p > B$ pivot substrings that are roughly evenly spaced. Then, we find the pivot that is lexicographically "closest" to the wanted k-th and one of the following two situations arises:

• We are able to infer that the kth smallest suffix is strictly between two consecutive pivots (that is its corresponding substring of B characters is strictly greater and smaller of the two pivots). In this case, we return all the $O(N/p) = O(N/B)$ suffixes that are contained between the two pivots.

• We can identify the suffixes that have the first B characters equal to those of the kth smallest suffix. We show that, in case they are still $\Omega(N/B)$ in number, they must satisfy some periodicity property, so that we can reduce them to just $O(N/B)$ with additional $O(N/B)$ block transfers.

[2]Given strings S and T, their longest common prefix $lcp(S, T)$ is longest string U such that both S and T start with U.

4.1. Finding pivots and the key suffixes

Let $p = \sqrt{\frac{M^c}{B}}$, for a suitable constant $c > 1$. We proceed with the following steps.

First. We sort the first M^c substrings of length B of S (that is substrings $S[1 \dots B]$, $S[2 \dots B+1]$,..., $S[M^c-1 \dots B+M^c-2]$, $S[M^c \dots B+M^c-1]$). Then we sort the second M^c substrings of length B and so forth until all the N positions in S have been considered. The product of this step is an array V of N pointers to the substrings of length B of S.

Second. We scan V and we collect in an array U of N/p positions the N/p pointers $V[p], V[2p], V[3p], \dots$.

Third. We (multi)-select from U the p pointers to the substrings (of length B) $b_1, \dots, b_p$ such that b_i has rank $i\frac{N}{p^2}$ among the substrings (pointed by the pointers) in U. These are the pivots we were looking for. We store the p (pointers to the) pivots in an array U'.

Fourth. We need to find the rightmost pivot b_x such that the number of substrings (of length B of S) lexicographically smaller than b_x is less than k (the rank of the wanted suffix). We cannot simply distribute all the substrings of length B according to all the p pivots in U', because it would be too costly. Instead, we proceed with the following refining strategy.

1. From the p pivots in U' we extract the group G_1 of δM equidistant pivots, where $\delta < 1$ is a suitable constant, (i.e. the pivots $b_t, b_{2t}, \dots$, where $t = \frac{p}{\delta M}$). Then, for any $b_j \in G_1$, we find out how many substrings of size B are lexicographically smaller than b_j. After that we find the rightmost pivot $b_{x_1} \in G_1$ such that the number of substrings (of length B) smaller than b_{x_1} is less than k.
2. From the $\frac{p}{\delta M}$ pivots in U' following b_{x_1} we extract the group G_2 of δM equidistant pivots. Then, for any $b_j \in G_2$, we find out how many substrings of size B are smaller than b_j. After that we find the rightmost pivot $b_{x_2} \in G_2$ such that the number of substrings smaller than b_{x_2} is less than k.

More generally:

f. Let G_f be the δM pivots in U' following $b_{x_{f-1}}$. Then, for any $b_j \in G_f$, we find out how many substrings of size B are smaller than b_j. After that we find the rightmost pivot $b_{x_f} \in G_f$ such that the number of substrings smaller than b_{x_f} is less than k.

The pivot b_{x_f} found in the last iteration is the pivot b_x we are looking for in this step.

Fifth. We scan S and compute the following two numbers: the number $n_x^<$ of substrings of length B lexicographically smaller than b_x; the number $n_x^=$ of substrings equal to b_x.

Sixth. In this step we treat the following case: $n_x^< < k \leq n_x^< + n_x^=$. More specifically, this implies that the wanted kth smallest suffix has its prefix of B characters equal to b_x. We proceed as follows. We scan S and gather in a contiguous zone R (the indexes of) the suffixes of S having their prefixes of B characters equal to b_x. In this case we have already found the key suffixes (whose indexes reside in R). Therefore the computation in this section ends here and we proceed to discard some of them (sec. 4.2).

Seventh. In this step we treat the following remaining case: $n_x^< + n_x^= < k$. In other words, in this case we know that the prefix of B characters of the wanted kth smallest suffix is (lexicographically) greater than b_x and smaller than b_{x+1}. Therefore, we scan S and gather in a contiguous zone R (the indexes of) the suffixes of S having their prefix of B characters greater than b_x and smaller than b_{x+1}. Since there are less than N/B such suffixes (see below Lemma 4.1), we have already found the set of sparse active suffixes (whose indexes reside in R) that will be processed in Sec. 3.

Lemma 4.1. *For any S and k, either the number of key suffixes found is $O(N/B)$, or their prefixes of B characters are all the same.*

Lemma 4.2. *Under the tall-cache assumption, finding the key suffixes needs $O(N/B)$ block transfers in the worst case.*

4.2. Discarding key suffixes

Finally, let us show how to reduce the number of key suffixes gathered in Sec. 4.1 to $\leq 2N/B$ so that we can pass them to the sparse suffix selection algorithm (Sec. 3). Let us assume that the number of key suffixes is greater than $2N/B$.

The indexes of the key suffixes have been previously stored in an array R. Clearly, the kth smallest suffix is among the ones in R. We also know the number $n^<$ of suffixes of S that are lexicographically smaller than each suffix in R. Finally, we know that there exists a string q of length B such that R *contains all and only the suffixes s_i such that the prefix of length B of s_i is equal to q* (i.e. R contains the indexes of all the occurrences of q in S).

To achieve our goal we exploit the possible periodicity of the string q. A string u is *a period* of a string v ($|u| \leq |v|$) if v is a prefix of u^i for some integer $i \geq 1$. *The period* of v is the smallest of its periods. We exploit the following:

Property 1 ([8])**.** If q occurs in two positions i and j of S and $0 < j - i < |q|$ then q has a period of length $j - i$.

Let u be the period of q. Since the number of suffixes in R is greater than $2N/B$, *there must be some overlapping* between the occurrences of q in S. Therefore, by Property 1, we can conclude that $|u| < |q|$. For the sake of presentation let us assume that $|q|$ *is not a multiple of* $|u|$ (the other case is analogous).

From how R has been built (by left to right scanning of S) we know that the indexes in it are in increasing order, that is $R[i] < R[i+1]$, for any i (i.e. the indexes in R follow the order, from left to right, in which the corresponding suffixes may be found in S). Let us consider a *maximal subsequence* R_i of R such that, for any $1 \leq j < |R_i|$, $R_i[j+1] - R_i[j] \leq B/2$ (i.e. the occurrence of q in S starting in position $R_i[j]$ overlaps the one starting in position $R_i[j+1]$ by at least $B/2$ positions). Clearly, any two of these subsequences of R do not overlap and hence R can be seen as the concatenation $R_1 R_2 \cdots$ of these subsequences. From the definition of the partitioning of R and from the periodicity of q we have:

Lemma 4.3. *The following statements hold:*

(i) *There are less than $2N/B$ such subsequences.*
(ii) *For any R_i, the substring $S[R_i[1] \dots R_i[|R_i|] + B - 1]$ (the substring of S spanned by the substrings whose indexes are in R_i) has period u.*
(iii) *The substring of length B of S starting in position $R_i[|R_i|] + |u|$ (the substring starting one period-length past the rightmost member of R_i) is not equal to q.*

For any key suffix s_j, let us consider the following prefix: $ps_j = S[j \dots R_i[|R_i|] + |u| + B - 1]$, where R_i is the subsequence of R where (the index of) s_j belongs to. By Lemma 4.3, we know two things about ps_j: (a) the prefix of length $|ps_j| - |u|$ of ps_j has period u; (b) the suffix of length B of ps_j *is not equal* to q.

In light of this, we associate with any key suffix s_j a pair of integers $\langle \alpha_j, \beta_j \rangle$ defined as follows: α_j is equal to *the number of complete periods* u in the prefix of length $|ps_j| - |u|$ of ps_j; β_j is equal to $|R_i| + |u|$ (that is the index of the substring of length B starting one period-length past the rightmost member of R_i).

There is natural total order $\lhd$ that can be defined over the key suffixes. It is based on the pairs of integers $\langle \alpha_j, \beta_j \rangle$ and it is defined as follow. For any two key suffixes $s_{j'}, s_{j''}$:

- If $\alpha_{j'} = \alpha_{j''}$ then $s_{j'}$ and $s_{j''}$ are equal (according to $\lhd$).
- If $\alpha_{j'} < \alpha_{j''}$ then $s_{j'} \lhd s_{j''}$ iff $S[\beta_{j'} \dots \beta_{j'} + B - 1]$ is lexicographically smaller than q.

By Lemma 4.3, we know that the suffix of length B of $ps_{j'}$ (which is the substring $S[\beta_{j'} \dots \beta_{j'} + B - 1]$) *is not equal* to q. Therefore the total order $\lhd$ is well defined.

We are now ready to describe the process for reducing the number of key suffixes. We proceed with the following steps.

First. By scanning S and R, we compute the pair $\langle \alpha_j, \beta_j \rangle$ for any key suffix s_j. The pairs are stored in an array (of pairs of integers) $Pairs$.

Second. We scan S and compute the array $Comp$ of N positions defined as follows: for any $1 \leq i \leq N$, $Comp[i]$ is equal to -1, 0 or 1 if $S[i \dots i + B - 1]$ is less than, equal to or greater than q, respectively (the array $Comp$ tells us what is the result of the comparison of q with any substring of size B different from it).

Third. By scanning $Pairs$ and $Comp$ at the same time, we compute the array $PComp$ of size $|Pairs|$, such that, for any l, $PComp[l] = Comp[Pairs[l].\beta]]$ (where $Pairs[l].\beta$ is the second member of the pair of integers in position l of $Pairs$).

Fourth. Using $Pairs$ and $PComp$, we select the $(k - n^<)$-th smallest key suffix s_x and all the key suffixes equal to s_x *according to the total order* $\lhd$ (where $n^<$ is the number of suffixes of S that are lexicographically smaller than each suffix in R, known since Sec. 4.1). The set of the selected key suffixes is the output of the process.

Lemma 4.4. *At the end of the discarding process, the selected key suffixes are less than* $2N/B$ *in number and the* k*th lexicographically smallest suffix is among them.*

Lemma 4.5. *The discarding process requires* $O(N/B)$ *block transfers at the worst case.*

Theorem 4.6. *The suffix selection problem for a string defined over a general alphabet can be solved using* $O(N/B)$ *block transfers in the worst case.*

References

[1] A. Aggarwal and J. Vitter. The input/output complexity of sorting and related problems. In *Communications of ACM*, 1988.
[2] M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. System Sci.*, 7:448–61, 1973.
[3] David Dobkin and J. Ian Munro. Optimal time minimal space selection algorithms. *Journal of the ACM*, 28(3):454–461, July 1981.
[4] M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. 38th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 137–143. IEEE, 1997.
[5] M. Farach, P. Ferragina, and S. Muthukrishnan. Overcoming the memory bottleneck in suffix tree construction. In *Proc. 39th Annual Symp. on Foundations of Computer Science (FOCS)*. IEEE, 1998.
[6] G. Franceschini and S. Muthukrishnan. Optimal suffix selection. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC)*, 2007.
[7] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Annual Symp. on Foundations of Computer Science (FOCS 1999)*, pages 285–297. IEEE, 1999.
[8] Z. Galil. Optimal parallel algorithms for string matching. *Inf. Control*, 67(1-3):144–157, 1985.
[9] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.
[10] J. Vitter. In *Algorithms and Data Structures for External Memory*, 2007.
[11] P. Weiner. Linear pattern matching algorithms. In *Foundations of Computer Science (FOCS)*, 1973.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 469–480
www.stacs-conf.org

RANDOMNESS ON COMPUTABLE PROBABILITY SPACES—A DYNAMICAL POINT OF VIEW

PETER GÁCS [1] AND MATHIEU HOYRUP [2] AND CRISTÓBAL ROJAS [3]

[1] Computer Science Department, Boston University

[2] Département d'Informatique, École Normale Supérieure de Paris

[3] DI École Normale Supérieure and CREA École Polytechnique, Paris.

Abstract. We extend the notion of randomness (in the version introduced by Schnorr) to computable Probability Spaces and compare it to a *dynamical* notion of randomness: typicality. Roughly, a point is *typical* for some dynamic, if it follows the statistical behavior of the system (Birkhoff's pointwise ergodic theorem). We prove that a point is Schnorr random if and only if it is typical for every *mixing* computable dynamics. To prove the result we develop some tools for the theory of computable probability spaces (for example, morphisms) that are expected to have other applications.

1. Introduction

The roots of algorithmic randomness go back to the work of von Mises in the 20th century. He suggested a notion of individual infinite random sequence based on *limit-frequency* properties invariant under the action of *selection* functions from some "acceptable" set. The problem was then to properly define what an "acceptable" selection function could be. Some years later, the concept of *computable* function was formalized, providing a natural class of functions to be considered as acceptable. This gave rise to Church's notion of *computable randomness*. Nevertheless, substantial understanding was achieved only with the works of Kolmogorov [7], Martin-Löf [8], Levin [17] and Schnorr [9] and since then, many efforts have contributed to the development of this theory which is now well established and intensively studied.

There are several different possible definitions, but it is Martin-Löf's one which has received most attention. This notion can be defined, at least, from three different points of view:

(1) *measure theoretic*. This was the original presentation by Martin-Löf ([8]). Roughly, an infinite sequence is random if it satisfies all "effective" probabilistic laws (see definition 3.21).

1998 ACM Subject Classification: Theory of Computation (F.0), Probability and Statistics (G.3), Information Theory (H.1.1).

Key words and phrases: Schnorr Randomness, Birkhoff's ergodic theorem, computable measures.

PARTLY SUPPORTED BY ANR GRANT 05 2452 260 OX

(2) *compressibility.* This characterization of random sequences, due to Schnorr and Levin (see [17, 10]), uses the prefix-free Kolmogorov complexity: random sequences are those which are maximally complex.
(3) *predictability.* In this approach (started by Ville [13] and reintroduced to the modern theory by Schnorr [10]) a sequence is random if, in a fair betting game, no "effective" strategy ("martingale") can win an unbounded amount of money against it.

In [9], a somewhat broader notion of algorithmic randomness (narrower notion of probabilistic law) was proposed: Schnorr randomness. This notion received less attention over the years: Martin-Löfs definition is simpler, leads to universal tests, and many equivalent characterizations (besides, Schnorr's book is not in English...). Recently, Schnorr randomness has begun to receive more attention. The work [2] for instance, characterizes it in terms of Kolmogorov complexity.

In the present paper, first we extend Schnorr randomness to arbitrary computable probability spaces and develop some useful tools. Then, taking a *dynamical systems* point of view, we introduce yet another approach to the definition of randomness: typicality. Roughly, a point is *typical* for some measure-preserving ergodic dynamic, if it follows the statistical behavior of the system (given by Birkhoff's pointwise ergodic theorem) with respect to every bounded continous function used to follow its trajectory (or equivalently, every computable function, see Definition 3.28). We then show that:

Theorem. *In any computable probability space, a point is Schnorr random if and only if it is typical for every* mixing *computable dynamical system.*

The paper is organized as follows: Section 2 presents all needed concepts of computability theory and computable measure theory over general metric spaces. Parts of this section, for example on μ-computable functions, are new and should be of independent interest. Section 3.1 generalizes Schnorr randomness and studies some useful properties, after which we introduce the notion of typicality. Section 3.3 is devoted to the proof of our main result.

2. Computability

In classical recursion theory, a set of natural numbers is called ***recursively enumerable*** (***r.e.*** for short) if it is the range of some partial recursive function. That is if there exists an algorithm listing (or enumerating) the set.

Strictly speaking, recursive functions only work on natural numbers, but this can be extended to the objects (thought of as "finite" objects) of any countable set, once a numbering of its elements has been chosen. We will sometimes use the word ***algorithm*** instead of *recursive function* when the inputs or outputs are interpreted as finite objects. The operative power of an algorithm on the objects of such a numbered set obviously depends on what can be effectively recovered from their numbers.

Examples 2.1.
1 $\mathbb{N}^k$ can be numbered in such a way that the k-tuple of number i can be computed from i and vice versa.
2 The set $\mathbb{Q}$ of rational numbers can be injectively numbered $\mathbb{Q} = \{q_0, q_1, \ldots\}$ in an *effective* way: the number i of a rational a/b can be computed from a and b, and vice versa. We fix such a numbering.

All through this work, we will use recursive functions over numbered sets to define *computability* or *constructivity* notions on *infinite* objects. Depending on the context, these notions will take particulars names (computable, recursively enumerable, r.e. open, decidable, etc...) but the definition will be always of the form: *obect x is* **constructive** *if there exists a recursive $\varphi\colon \mathbb{N} \to D$ satisfying* property *$P(\varphi, x)$* (where D is some numbered set).

For example, *$E \subset \mathbb{N}$ is* **r.e.** *if there exists a recursive $\varphi\colon \mathbb{N} \to \mathbb{N}$ satisfying $E =$* range(φ).

Each time, a *uniform version* will be implicitly defined: *a sequence $(x_i)_i$ is constructive* **uniformly in *i*** *if there exists a recursive $\varphi\colon \mathbb{N} \times \mathbb{N} \to D$ satisfying* property *$P(\varphi(i, \cdot), x_i)$ for all i.*

In our example, *a sequence $(E_i)_i$ is* **r.e. uniformly in *i*** *if there exists $\varphi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ satisfying $E_i =$* range$(\varphi(i, \cdot))$ *for all i.*

Let us ilustrate this in the case of reals numbers (computable reals numbers were introduced by Turing in [11]).

Definition 2.2. A real number $x \in \mathbb{R}$ is said to be ***computable*** if there exists a total recursive $\varphi : \mathbb{N} \to \mathbb{Q}$ satisfying $|x - \varphi(n)| < 2^{-n}$ for all $n \in \mathbb{N}$.

Hence by a sequence of reals $(x_i)_i$ computable ***uniformly in i*** we mean that there exists a recursive $\varphi : \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ satisfying $|x - \varphi(i, n)| < 2^{-n}$ for all $n \in \mathbb{N}$, for all $i \in \mathbb{N}$.

We also have the following notions:

Definition 2.3. Let x be a real number. We say that:
- x is ***lower semi-computable*** if the set $\{i \in \mathbb{N} : q_i < x\}$ is r.e.,
- x is ***upper semi-computable*** if the set $\{i \in \mathbb{N} : q_i > x\}$ is r.e.,

It is easy to see that a real number is computable if and only if it is lower and upper semi-computable.

2.1. Computable metric spaces

We breifly recall the basic of computable metric spaces.

Definition 2.4. A ***computable metric space*** (CMS) is a triple $\mathcal{X} = (X, d, \mathcal{S})$, where
- (X, d) is a separable complete metric space.
- $\mathcal{S} = (s_i)_{i \in \mathbb{N}}$ is a numbered dense subset of X (called ***ideal points***).
- The real numbers $(d(s_i, s_j))_{i,j}$ are all computable, uniformly in i, j.

Some important examples of computable metric spaces:

Examples 2.5.
1 The Cantor space $(\Sigma^{\mathbb{N}}, d, S)$ with Σ a finite alphabet. If $x = x_1 x_2 \ldots$, $y = y_1 y_2 \ldots$, are elements then the distance is defined by $d(x, y) = \sum_{i : x_i \neq y_i} 2^{-i}$. Let us fix some element of Σ denoting it by 0. The dense set S is the set of ultimately 0-stationary sequences.
2 $(\mathbb{R}^n, d_{\mathbb{R}^n}, \mathbb{Q}^n)$ with the Euclidean metric and the standard numbering of $\mathbb{Q}^n$.

For further examples we refer to [15].

The numbered set of ideal points $(s_i)_i$ induces the numbered set of ***ideal balls*** $\mathcal{B} := \{B(s_i, q_j) : s_i \in S, q_j \in \mathbb{Q}_{>0}\}$. We denote by $B_{\langle i,j \rangle}$ (or just B_n) the ideal ball $B(s_i, q_j)$, where $\langle \cdot, \cdot \rangle$ is a computable bijection between tuples and integers.

Definition 2.6 (Computable points)**.** A point $x \in X$ is said to be ***computable*** if the set $E_x := \{i \in \mathbb{N} : x \in B_i\}$ is r.e.

Definition 2.7 (R.e. open sets)**.** We say that the set $U \subset X$ is ***r.e. open*** if there is some r.e. set $E \subset \mathbb{N}$ such that $U = \bigcup_{i \in E} B_i$. If U is r.e. open and $D \subset X$ is an arbitrary set then the set $A := U \cap D$ is called ***r.e. open in D***.

Examples 2.8.
1 If the sequence $(U_n)_n$ is r.e. open uniformly in n, then the union $\bigcup_n U_n$ is an r.e. open set.
2 $U_i \cup U_j$ and $U_i \cap U_j$ are r.e. open uniformly in (i, j). See [5].

Let (X, S_X, d_X) and (Y, S_Y, d_Y) be computable metric spaces. Let $(B_i^Y)_i$ be the collection of ideal balls from Y.

Definition 2.9 (Computable Functions)**.** A function $T : X \to Y$ is said to be ***computable*** if $T^{-1}(B_i^Y)$ is r.e. open uniformly in i.

It follows that computable functions are continuous. Since we will work with functions which are not necessarily continuous everywhere (and hence not computable), we shall consider functions which are computable on some subset of X. More precisely, a function T is said to be ***computable on D*** ($D \subset X$) if $T^{-1}(B_i^Y)$ is r.e. open in D, uniformly in i. The set D is called the ***domain of computability*** of T.

3. Computable Probability Spaces

Let us recall some basic concepts of measure theory. Let X be a set. A family $\mathfrak{B}$ of subsets of X is called an ***algebra*** if (i)$X \in \mathfrak{B}$, (ii)$A \in \mathfrak{B} \Rightarrow A^{\mathcal{C}} \in \mathfrak{B}$ and (iii) $A, B \in \mathfrak{B} \Rightarrow A \cup B \in \mathfrak{B}$. We say that $\mathfrak{B}$ is a ***σ-algebra*** if moreover $A_i \in \mathfrak{B}, i \geq 1 \Rightarrow \bigcup_i A_i \in \mathfrak{B}$. If $\mathfrak{B}_0$ is a family of subsets of X, the σ-algebra generated by $\mathfrak{B}_0$ (denoted $\sigma(\mathfrak{B}_0)$) is defined to be the smallest σ-algebra over X that contains $\mathfrak{B}_0$. If $\mathfrak{B}$ is a σ-algebra of subsets of X, we say that $\mu : \mathfrak{B} \to [0, 1]$ is a ***probability measure*** if, for every family $(A_i)_i \subset \mathfrak{B}$ of disjoint subsets of X, the following holds:

$$\mu\Big(\bigcup_i A_i\Big) = \sum_i \mu(A_i). \tag{3.1}$$

If X is a topological space, the ***Borel*** σ-algebra of X is defined as the σ-algebra generated by the family of open sets of X. Sets in the Borel σ-algebra are called Borel sets. In this paper, a ***probability space*** will always refer to the triple $(X, \mathfrak{B}, \mu)$, where $\mathfrak{B}$ is the Borel σ-algebra of X and μ is a probability measure. A set $A \subset X$ has ***measure zero*** if there is a Borel set A_1 such that $A \subset A_1$ and $\mu(A_1) = 0$. We call two sets $A_1, A_2 \subset X$ ***equivalent modulo zero***, and write $A_1 = A_2 \pmod 0$, if the symmetric difference has measure zero. We write $A_1 \subset A_2 \pmod 0$ if A_1 is a subset of A_2 and $A_1 = A_2 \pmod 0$.

When X is a computable metric space, the space of probability measures over X, denoted by $\mathcal{M}(X)$, can be endowed with a structure of computable metric space. Then a computable measure can be defined as a computable point in $\mathcal{M}(X)$.

Example 3.1 (Measure over a Cantor space)**.** As a special example, we can set $X = \mathbb{B}^{\mathbb{N}}$ where $\mathbb{B} = \{0, 1\}$ and $\lambda([x]) = 2^{-|x|}$, where $|x|$ is the length of the binary string $x \in \{0, 1\}^*$.

This is the distribution on the set of infinite binary sequences obtained by tossing a fair coin, and condition (3.1) simplifies to

$$\lambda(x0) + \lambda(x1) = \lambda(x).$$

Let $\mathcal{X} = (X, d, S)$ be a computable metric space. Let us consider the space $\mathcal{M}(X)$ of measures over X endowed with weak topology, that is:

$$\mu_n \to \mu \text{ iff } \mu_n f \to \mu f \text{ for all real continuous bounded } f,$$

where μf stands for $\int f \, \mathrm{d}\mu$.

If X is separable and complete, then $\mathcal{M}(X)$ is separable and complete. Let $D \subset \mathcal{M}(X)$ be the set of those probability measures that are concentrated in finitely many points of S and assign rational values to them. It can be shown that this is a dense subset ([1]).

We consider the Prokhorov metric ρ on $\mathcal{M}(X)$ defined by:

$$\rho(\mu, \nu) := \inf\{\epsilon \in \mathbb{R}^+ : \mu(A) \leq \nu(A^\epsilon) + \epsilon \text{ for every Borel set } A\}$$

where $A^\epsilon = \{x : d(x, A) < \epsilon\}$.

This metric induces the weak topology on $\mathcal{M}(X)$. Furthermore, it can be shown that the triple $(\mathcal{M}(X), D, \rho)$ is a computable metric space (see [3], [5]).

Definition 3.2. A measure μ is computable if it is a computable point of $(\mathcal{M}(X), D, \rho)$

The following result (see [5]) will be intensively used in the sequel:

Lemma 3.3. *A probability measure μ is computable if and only if the measure of finite union of ideal balls $\mu(B_{i_1} \cup \ldots \cup B_{i_k})$ is lower semi-computable, uniformly in $i_1, \ldots, i_k$.*

Definition 3.4. A ***computable probability space (CPS)*** is a pair $(\mathcal{X}, \mu)$ where $\mathcal{X}$ is a computable metric space and μ is a computable Borel probability measure on X.

As already said, a computable function defined on the whole space is necessarily continuous. But a transformation or an observable need not be continuous at every point, as many interesting examples prove (piecewise-defined transformations, characteristic functions of measurable sets,...), so the requirement of being computable everywhere is too strong. In a measure-theoretical setting, the natural weaker condition is to require the function to be computable *almost everywhere*. In the computable setting this is not enough, and a computable condition on the set on which the function is computable is needed:

Definition 3.5 (Constructive G_δ-sets)**.** We say that the set $D \subset X$ is a ***constructive G_δ-set*** if it is the intersection of a sequence of uniformly r.e. open sets.

Definition 3.6 (μ-computable functions)**.** Let $(\mathcal{X}, \mu)$ and $\mathcal{Y}$ be a CPS and a CMS respectively. A function $f : (\mathcal{X}, \mu) \to Y$ is ***μ-computable*** if it is computable on a constructive G_δ-set (denoted as $\mathrm{dom} f$ or D_f) of measure one.

Example 3.7. Let m be the Lebesgue measure on $[0, 1]$. The binary expansion of reals defines a function from non-dyadic numbers to infinite binary sequences which induces a m-computable function from $([0, 1], m)$ to $\{0, 1\}^{\mathbb{N}}$.

Remark 3.8. Given a uniform sequence of μ-computable functions $(f_i)_i$, any computable operation $\odot_{i=0}^n f_i$ (adition, multiplication, composition, etc...) is μ-computable, uniformly in n.

We recall that $F : (\mathcal{X}, \mu) \to (\mathcal{Y}, \nu)$ is measure-preserving if $\mu(F^{-1}(A)) = \nu(A)$ for all Borel sets A.

Definition 3.9 (morphisms of CPS's)**.** A ***morphism of CPS's*** $F : (\mathcal{X}, \mu) \to (\mathcal{Y}, \nu)$, is a μ-computable measure-preserving function $F : D_F \subseteq X \to Y$.

An ***isomorphism of CPS's*** $(F, G) : (\mathcal{X}, \mu) \rightleftarrows (\mathcal{Y}, \nu)$ is a pair (F, G) of morphisms such that $G \circ F = \mathrm{id}$ on $F^{-1}(D_G)$ and $F \circ G = \mathrm{id}$ on $G^{-1}(D_F)$.

Example 3.10. Let $(\mathbb{B}^{\mathbb{N}}, \lambda)$ the probability space introduced in Example 3.1 with the coin-tossing distribution λ over the infinite sequences. The binary expansion (see example 3.7) creates an isomorphism of CPS's between the spaces $([0, 1], m)$ and $(\mathbb{B}^{\mathbb{N}}, \lambda)$.

Remark 3.11. To every isomorphism of CPS's (F, G) one can associate the canonical invertible morphism of CPS's $\varphi = F|_{D_\varphi}$ with $\varphi^{-1} = G|_{D_{\varphi^{-1}}}$, where $D_\varphi = F^{-1}(G^{-1}(D_F))$ and $D_{\varphi^{-1}} = G^{-1}(D_F)$. Of course, (φ, φ^{-1}) is an isomorphism of CPS's as well.

The next proposition is a direct consequence of theorem 5.1.1 from [5]:

Proposition 3.12. *Every computable probability space is isomorphic to the Cantor space with an appropiate computable measure.*

Definition 3.13. A set $A \subset X$ is said to be ***almost decidable*** if the function $1_A : X \to \{0, 1\}$ is μ-computable.

It is easy to see that a set A is almost decidable iff there is a constructive G_δ set D of measure one and two r.e. open sets U and V such that:

$$U \cap D \subset A, \quad V \cap D \subseteq A^{\mathcal{C}}, \quad \mu(U) + \mu(V) = 1.$$

Remarks 3.14.

1 The collection of almost decidable sets is an algebra.
2 An almost decidable set is always a continuity set.
3 Ideal balls with zero boundary measure are always almost decidable.
4 Unless the space is disconnected (i.e. has non-trivial clopen subsets), no set can be *decidable*, i.e. semi-decidable (r.e.) and with a semi-decidable complement (such a set must be clopen[1]). Instead, a set can be decidable *with probability* 1: there is an algorithm which decides if a point belongs to the set or not, for almost every point. This is why we call it *almost decidable*.

Ignoring computability, the existence of open sets with zero boundary measure directly follows from the fact that the collection of open sets is uncountable and μ is finite. The problem in the computable setting is that there are only countable many open r.e. sets. Fortunately, there still always exists a basis of almost decidables balls.

Lemma 3.15. *Let X be $\mathbb{R}$ or $\mathbb{R}^+$ or $[0, 1]$. Let μ be a computable probability measure on X. Then there is a sequence of uniformly computable reals $(x_n)_n$ which is dense in X and such that $\mu(\{x_n\}) = 0$ for all n.*

[1] In the Cantor space for example (which is totally disconnected), every cylinder (ball) is a decidable set. Indeed, to decide if some infinite sequence belongs to some cylinder it suffices to compare the finite word defining the cylinder to the corresponding finite prefix of the infinite sequence.

Proof. Let I be a closed rational interval. We construct $x \in I$ such that $\mu(\{x\}) = 0$. To do this, we construct inductively a nested sequence of closed intervals J_k of measure $< 2^{-k+1}$, with $J_0 = I$. Suppose $J_k = [a, b]$ has been constructed, with $\mu(J_k) < 2^{-k+1}$. Let $m = (b - a)/3$: one of the intervals $[a, a + m]$ and $[b - m, b]$ must have measure $< 2^{-k}$, and since their measure is upper-computable, we can find it effectively—let it be J_{k+1}.

From a constructive enumeration $(I_n)_n$ of all the dyadic intervals, we can construct $x_n \in I_n$ uniformly. ∎

Corollary 3.16. *Let $(\mathcal{X}, \mu)$ be a CPS and $(f_i)_i$ be a sequence of uniformly computable real valued functions on X. Then there is a sequence of uniformly computable reals $(x_n)_n$ which is dense in $\mathbb{R}$ and such that $\mu(\{f_i^{-1}(x_n)\}) = 0$ for all i, n.*

Proof. Consider the uniformly computable measures $\mu_i = \mu \circ f_i^{-1}$ and define $\nu = \sum_i 2^{-i}\mu_i$. By Lemma 3.3, ν is a computable measure and then, by Lemma 3.15, there is a sequence of uniformly computable reals $(x_n)_n$ which is dense in $\mathbb{R}$ and such that $\nu(\{x_n\}) = 0$ for all n. Since $\nu(A) = 0$ iff $\mu_i(A) = 0$ for all i, we get $\mu(\{f_i^{-1}(x_n)\}) = 0$ for all i, n. ∎

The following result will be used many times in the sequel.

Corollary 3.17. *There is a sequence of uniformly computable reals $(r_n)_{n\in\mathbb{N}}$ such that $(B(s_i, r_n))_{i,n}$ is a basis of almost decidable balls.*

Proof. Apply Corollary 3.16 to $(f_i)_i$ defined by $f_i(x) = d(s_i, x)$. ∎

We remark that every ideal ball can be expressed as a r.e. union of almost decidable balls, and vice-versa. So the two bases are constructively equivalent.

Definition 3.18. A computable probability space is a ***computable Lebesgue space*** if it is isomorphic to the computable probability space $([0, 1], m)$ where m is the Lebesgue measure.

Theorem 3.19. *Every computable probability space with no atoms is a computable Lebesgue space.*

Proof. We first prove the result for $I = ([0, 1], \mu)$.

Lemma 3.20. *The interval endowed with a non-atomic computable probability measure is a computable Lebesgue space.*

Proof. We define the morphism of the CPS as $F(x) = \mu([0, x])$. As μ has no atom and is computable, F is computable and surjective. As F is surjective, it has right inverses. Two of them are $G_<(y) = \sup\{x : F(x) < y\}$ and $G_>(y) = \inf\{x : F(x) > y\}$, and satisfy $F^{-1}(\{y\}) = [G_<(y), G_>(y)]$. They are increasing and respectively left- and right-continuous. As F is computable, they are even lower- and upper semi-computable respectively. Let us define $D = \{y : G_<(y) = G_>(y)\}$: every $y \in D$ has a unique pre-image by F, which is then injective on $F^{-1}(D)$. The restriction of F on $F^{-1}(D)$ has a left-inverse, which is given by the restriction of $G_<$ and $G_>$ on D. Let us call it $G : D \to I$. By lower and upper semi-computability of $G_<$ and $G_>$, G is computable. Now, D is a constructive G_δ-set: $D = \bigcap_n \{y : G_>(y) - G_<(y) < 1/n\}$. We show that $I \setminus D$ is a countable set. The family $\{[G_<(y), G_>(y)] : y \in I\}$ indexed by I is a family of disjoint closed intervals, included in $[0, 1]$. Hence, only countably many of them have positive length. Those intervals correspond to points y belonging to $I \setminus D$, which is then countable. It follows that D has Lebesgue measure one (it is even dense). (F, G) is then an isomorphism between (I, μ) and (I, m). ∎

Now, we know from Theorem 3.12 that every CPS $(\mathcal{X}, \mu)$ has a binary representation, which is in particular an isomorphism with the Cantor space $(\mathbb{B}^{\mathbb{N}}, \mu')$. As mentioned in Example 3.10, the latter is isomorphic to (I, μ_I) where μ_I is the induced measure. If μ is non-atomic, so is μ_I. By the previous lemma, (I, μ_I) is isomorphic to (I, m). ∎

3.1. Randomness and typicality

3.1.1. *Algorithmic randomness.*

Definition 3.21. A ***Martin-Löf test*** (ML-test) is an uniform sequence $(A_n)_n$ of r.e. open sets such that $\mu(A_n) \leq 2^{-n}$. We say that x ***fails*** the ML-test if $x \in A_n$ for all n. A point x is called ***ML-random*** if it fails no ML-test.

Definition 3.22. A ***Borel-cantelli test*** (BC-test) is a uniform sequence $(C_n)_n$ of r.e. open sets such that $\sum_n \mu(C_n) < \infty$. We say that x ***fails*** the BC-test if $x \in C_n$ infinitly often (i.o.).

It is easy to show that:

Proposition 3.23. *x fails a ML-test iff x fails a* BC*-test.*

Definition 3.24. A ***Schnorr test*** (Sch-test) is a ML-test $(A_n)_n$ such that the sequence of reals $(\mu(A_n))_n$ is uniformly computable. We say that x fails the Sch-test if $x \in A_n$ for all n. A point x is called Sch***-random*** if it fails no Sch-test.

Definition 3.25. A ***strong*** BC***-test*** is a BC-test $(C_n)_n$ such that $\sum_n \mu(C_n)$ is computable.

Proposition 3.26. *An element x fails a* Sch*-test if and only if x fails a strong* BC*-test.*

Proof. Let $(C_n)_n$ be a strong BC-test. Let c be such that $2^c > \sum_n \mu(C_n)$. Define the r.e. open set $A_k := \{x : |\{n : x \in C_n\}| \geq 2^{k+c}\}$. Then $\mu(A_k) < 2^{-k}$. Observe that A_k is the union of all the (2^{k+c})-intersections of C_n's. Since $\mu(C_k) = \sum_n \mu(C_n) - \sum_{n \neq k} \mu(C_n)$ and the C_n's are r.e. we have that $\mu(C_n)$ is computable (uniformly in n). We choose a basis $(B^i)_i$ of almost decidable balls to work with. Recall that finite unions or intersections of almost decidable sets are almost decidable too and that the measure of an almost decidable set is computable. Now we show that $\mu(A_k)$ is computable uniformly in k. Let $\epsilon > 0$ be rational. Let n_0 be such that $\sum_{n \geq n_0} \mu(C_n) < \frac{\epsilon}{2}$. Then $\mu(\bigcup_{n \geq n_0} C_n) < \frac{\epsilon}{2}$. For each C_n with $n < n_0$ we construct an almost decidable set $C_n^\epsilon \subset C_n$ (a finite union of almost decidable balls) such that $\mu(C_n) - \mu(C_n^\epsilon) < \frac{1}{n_0}\frac{\epsilon}{2}$. Then $\sum_{n<n_0}[\mu(C_n) - \mu(C_n^\epsilon] < \frac{\epsilon}{2}$. Define A_k^ϵ to be the union of the (2^{k+c})-intersections of the C_n^ϵ's for $n < n_0$. Then A_k^ϵ is almost decidable and then has a computable measure. Moreover $A_k \subset A_k^\epsilon \cup (\bigcup_{n \geq n_0} C_n) \cup (\bigcup_{n<n_0} C_n \setminus C_n^\epsilon)$, hence $\mu(A_k) - \mu(A_k^\epsilon) < \epsilon$. ∎

The following result is an easy modification of a result from [5], so we omit the proof.

Proposition 3.27. *Morphisms of computable probability spaces are defined (and computable) on Schnorr random points and preserve* Sch*-randomness.*

3.2. Dynamical systems and typicality

Let X be a metric space, let $T : X \mapsto X$ be a Borel map. Let μ be an invariant Borel measure on X, that is: $\mu(A) = \mu(T^{-1}(A))$ holds for each measurable set A. A set A is called T-invariant if $T^{-1}(A) = A$ modulo a set of measure 0. The system (T, μ) is said to be ergodic if each T-invariant set has total or null measure. In such systems the famous Birkhoff ergodic theorem says that time averages computed along μ-typical orbits coincide with space averages with respect to μ. More precisely, for any $f \in L^1(X)$ it holds

$$\lim_{n\to\infty} \frac{S_n^f(x)}{n} = \int f \, \mathrm{d}\mu, \tag{3.2}$$

for μ-almost each x, where $S_n^f = f + f \circ T + \ldots + f \circ T^{n-1}$.

If a point x satisfies equation (3.2) for a certain f, then we say that x is ***typical*** with respect to the ***observable*** f.

Definition 3.28. If x is typical w.r. to every bounded continuous function $f : X \to \mathbb{R}$, then we call it a T-***typical point***.

Remark 3.29. The proof of our main theorem will show as a side result that the definition would not change if we replaced "continuous" with "computable" in it.

In [14] is proved that ML-random infinite binary sequences are typical w.r. to any computable f. In [4], this is generalized via effective symbolic dynamics to computable probability spaces and μ-computable observables.

To have the result for Sch-random points it seems that a certain "mixing" property or "loss of memory" of the system has to be required. This is naturally expressed by means of the ***correlation functions***. For measureable functions f, g let

$$\begin{aligned} C(f,g) &= \mu(f \cdot g) - \mu f \cdot \mu g, \\ C_n(f,g) &= C(f \circ T^n, g). \end{aligned}$$

For events A, B with indicator functions $1_A, 1_B$ let

$$C_n(A,B) = C_n(1_A, 1_B),$$

which measures the dependence between the events A and B at times $n \gg 1$ and 0 respectively. Note that $C_n(A,B) = 0$ corresponds, in probabilistic terms, to $T^{-n}(A)$ and B being independent events.

Let us say that a family of Borel sets $\mathcal{E}$ is ***essential***, if for every open set U there is a sequence $(E_i)_i$ of borel sets in $\mathcal{E}$ such that $\cup_i E_i \subset U \pmod 0$ (see Section 3).

Definition 3.30. We say that a system (X, T, μ) is (polynomially) ***mixing*** if there is $\alpha > 0$ and an essential family $E = \{E_1, E_2, ...\}$ of almost decidable events such that for each i, j there is $c_{i,j} > 0$ computable in i, j such that

$$|C_n(E_i, E_j)| \leq \frac{c_{i,j}}{n^\alpha} \qquad \text{for all } n \geq 1.$$

We say that the system is ***independent*** if all correlation functions $C_n(E_i, E_j)$ are 0 for sufficiently large n.

Examples of non-mixing but still ergodic systems are given for instance by irrational circle rotations with the Lebesgue measure. Examples of mixing but not independent sytems are given by piecewise expandings maps or uniformly hyperbolic systems which have a

distinguished ergodic measure (called SRB measure and which is "physical" in some sense) with respect to which the correlations decay exponentially (see [12]). An example of a mixing system for which the decrease of correlations is only polynomial and not exponential, is given by the class of *Manneville-Pomeau* type maps (non uniformly expanding with an indifferent fixed point, see [6]). For a survey see [16].

3.3. Proof of the main result

Now we prove our main theorem.

Theorem 3.31. Let $(\mathcal{X}, \mu)$ be a computable probability space with no atoms The following properties of a point $x \in X$ are equivalent.

(i) x is Schnorr random.
(ii) x is T-typical for every mixing endomorphism T.
(iii) x is T-typical for every independent endomorphism T.

Remark 3.32. If the measure μ is atomic, it is easy to see that:

(1) $(\mathcal{X}, \mu)$ admits a mixing endomorphism if and only if $\mu = \delta_x$ for some x. In this case the theorem still holds, the only random point being x.
(2) $(\mathcal{X}, \mu)$ admits an ergodic endomorphism if and only if $\mu = \frac{1}{n}(\delta_{x_1} + ... + \delta_{x_n})$ (where $x_i \neq x_j$, for all $i \neq j$). In this case, a point x is Schnorr random if and only if it is typical for every ergodic endomorphism if and only if it is an atom.

Proof. Let us first prove a useful lemma. Let $E \subset X$ be a Borel set. Denote by 1_E its indicator function. The ergodic theorem says that the following equality holds for almost every point:

$$\lim_n \frac{1}{n} \sum_{i=0}^{n-1} 1_E \circ T^i(x) = \mu(E). \tag{3.3}$$

Lemma 3.33. *Let $\mathcal{E}$ be an essential family of events. If x satisfies equation (3.3) for all $E \in \mathcal{E}$ then x is a T-typical point.*

Proof. We have to show that equation (3.3) holds for any bounded continuous observable f. First, we extend equation (3.3) to every continuity open set C. Let $(E_i)_i$ be a sequence of elements of $\mathcal{E}$ such that $\bigcup_i E_i \subseteq \mathrm{Int}(C)$ and $\mu(\bigcup_i E_i) = \mu(C)$. Define $C_k = \bigcup_{i \leq k} E_i$. Then $\mu(C_k) \nearrow \mu(C)$. For all k:

$$\liminf_n \frac{1}{n} \sum_{i=0}^{n-1} 1_C \circ T^i(x) \geq \lim_n \frac{1}{n} \sum_{i=0}^{n-1} 1_{C_k} \circ T^i(x) = \mu(C_k)$$

so $\liminf_n \frac{1}{n} \sum_{i=0}^{n-1} 1_C \circ T^i(x) \geq \mu(C)$. Applying the same argument to $X \setminus C$ gives the result.

Now we extend the result to bounded continuous functions. Let f be continuous and bounded ($|f| < M$) and let $\epsilon > 0$ be a real number. Then, since the measure μ is finite, there exist real numbers $r_1, \ldots, r_k \in [-M, M]$ (with $r_1 = -M$ and $r_k = M$) such that $|r_{i+1} - r_i| < \epsilon$ for all $i = 1, \ldots, k-1$ and $\mu(f^{-1}(\{r_i\})) = 0$ for all $i = 1, \ldots, k$. It follows that for $i = 1, \ldots, k-1$ the sets $C_i = f^{-1}(]r_i, r_{i+1}[)$ are all continuity open sets.

Hence the function $f_\epsilon = \sum_{i=1}^{k-1} r_i 1_{C_i}$ satisfies $\|f - f_\epsilon\|_\infty \leq \epsilon$ and then the result follows by density. ∎

We are now able to prove that (i) ⇒ (ii).

Let $E \in \mathcal{E}$. Put $f = 1_E$. Observe that f is μ-computable. For $\delta > 0$, define the deviation sets:

$$A_n^f(\delta) = \left\{ x \in X : \left| \frac{S_n^f(x)}{n} - \int f \, \mathrm{d}\mu \right| > \delta \right\}.$$

By Corollary 3.16 we can choose δ such that $A_n^f(\delta)$ is almost decidable. Then their measures are computable, uniformly in n.

By the Chebychev inequality, $\mu(A_n^f(\delta)) \leq \frac{1}{\delta^2} \left\| \frac{S_n^f(x)}{n} - \int f \, \mathrm{d}\mu \right\|_{L^2}^2$. Let us change f by adding a constant to have $\int f \, \mathrm{d}\mu = 0$. This does not change the above quantity. Then, by invariance of μ we have

$$\left\| \frac{S_n^f(x)}{n} - \int f \, \mathrm{d}\mu \right\|_{L^2}^2 = \int \left(\frac{S_n^f(x)}{n} \right)^2 \mathrm{d}\mu = \frac{1}{n^2} \int n f^2 \, \mathrm{d}\mu + \frac{2}{n^2} \int \Big(\sum_{i<j<n} f \circ T^{j-i} f \Big) \, \mathrm{d}\mu$$

and hence

$$\delta^2 \mu(A_n^f(\delta)) \leq \frac{\|f\|_{L^2}^2}{n} + \frac{2}{n} \sum_{k<n} |C_k(f,f)| \leq \frac{\|f\|_{L^2}^2}{n} + \frac{2c_{f,f}}{(1-\alpha)n^\alpha}.$$

(Observe that α can be replaced by any smaller positive number, so we assume $\alpha < 1$.) Hence, $\mu(A_n^f(\delta)) \leq Cn^{-\alpha}$ for some constant C. Now, it is easy to find a sequence $(n_i)_{i\in\mathbb{N}}$ such that the subsequence $(n_i^{-\alpha})_i$ is effectively summable and $\frac{n_i}{n_{i+1}} \to 1$ (take for instance $n_i = i^\beta$ with $\alpha\beta > 1$). This shows that the sequence $A_{n_i}^f(\delta)$ is a strong BC-test. Therefore, if x is Sch-random then x belongs to only finitely many $A_{n_i}^f(\delta)$ for any δ and hence the subsequence $\frac{S_{n_i}^f(x)}{n_i}$ converges to $\int f \, \mathrm{d}\mu = \mu(E)$. To show that for such points the whole sequence $\frac{S_n^f(x)}{n}$ converges to $\int f \, \mathrm{d}\mu = \mu(E)$, observe that if $n_i \leq n < n_{i+1}$ and $\beta_i := \frac{n_i}{n_{i+1}}$ then we have:

$$\frac{S_{n_i}^f}{n_i} - 2(1-\beta_i)M \leq \frac{S_n^f}{n} \leq \frac{S_{n_{i+1}}^f}{n_{i+1}} + 2(1-\beta_i)M,$$

where M is a bound of f. To see this, for any k, l, β with $\beta \leq k/l \leq 1$:

$$\frac{S_k^f}{k} - \frac{S_l^f}{l} = \left(1 - \frac{k}{l}\right) \frac{S_k^f}{k} - \frac{S_{l-k}^f \circ T^{l-k}}{l} \leq (1-\beta)M + \frac{(l-k)M}{l} = 2(1-\beta)M.$$

Taking $\beta = \beta_i$ and $k = n_i$, $l = n$ first and then $k = n$, $l = n_{i+1}$ gives the result. Thus, we have proved that a Schnorr random point x satisfies equation (3.3) for any $E \in \mathcal{E}$. Lemma 3.33 allows to conclude.

The (ii) ⇒ (iii) part follows since any independent dynamic is in particular mixing.

To prove the (iii) ⇒ (i) part we will need the following proposition which is a strengthening of a result of Schnorr in [9]. The proof is somewhat technical, for lack of space we do not included here (see appendix).

Proposition 3.34. *If the infinite binary string $\omega \in (\mathbb{B}^\mathbb{N}, \lambda)$ is not Schnorr random (w.r. to the uniform measure), then there exists an isomorphism $\Phi : (\mathbb{B}^\mathbb{N}, \lambda) \to (\mathbb{B}^\mathbb{N}, \lambda)$ such that $\Phi(\omega)$ is not typical for the shift transformation σ.*

Now we are able to finish the proof of our main result: suppose that x is not Schnorr random. We construct a dynamic T for which x is not T-typical. From Proposition 3.12 and Theorem 3.19 we know that there is an isomorphism $\eta : (\mathcal{X}, \mu) \to (\mathbb{B}^{\mathbb{N}}, \lambda)$ (here, λ denotes the uniform measure). If $x \notin \mathrm{dom}(\eta)$, we can take any independent endomorphism and modify it in order to be the identity on x. It is cleary still an independent endomorphism (maybe with a smaller domain of computability) and x, being a fixed point, can't be T-typical. So let $x \in \mathrm{dom}(\eta)$. Then $\eta(x)$ is not Schnorr random in $(\mathbb{B}^{\mathbb{N}}, \lambda)$, since η as well as its inverse preserve Schnorr randomness. Then, by Proposition 3.34, $\Phi(\eta(x))$ is not σ-typical, where σ is the shift which is clearly independent (cylinders being the essential events). Put $\psi = \Phi \circ \eta$. Define the dynamics T on X by $T = \psi^{-1} \circ \sigma \circ \psi$. It is easy to see that T is independent for events of the form $E = \psi^{-1}[w]$. Since $\{\psi^{-1}[w] : w \in 2^*\}$ form an essential family of almost decidable events, T is independent too. As $\psi(x)$ is not σ-typical, x is not T-typical either. ∎

References

[1] Patrick Billingsley. *Convergence of Probability Measures*. John Wiley, New York, 1968.

[2] Rodney G. Downey and Evan J. Griffiths. Schnorr randomness. *Electr. Notes Theor. Comput. Sci.*, 66(1), 2002.

[3] Peter Gács. Uniform test of algorithmic randomness over a general space. *Theoretical Computer Science*, 341:91–137, 2005.

[4] Stefano Galatolo, Mathieu Hoyrup, and Cristóbal Rojas. Effective symbolic dynamics, random points, statistical behavior, complexity and entropy. *Submitted*, 2007.

[5] Mathieu Hoyrup and Cristóbal Rojas. Computability of probability measures and Martin-Löf randomness over metric spaces. *Information and Computation*, 2009. To appear.

[6] Stefano Isola. On systems with finite ergodic degree. *Far east journal of dynamical systems*, 5:1, 2003.

[7] Andrey N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems in Information Transmission*, 1:1–7, 1965.

[8] Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966.

[9] Claus-Peter Schnorr. *Zufälligkeit und Wahrscheinlichkeit. Eine algorithmische Begründung der Wahrscheinlichkeitstheorie*, volume 218 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin-New York, 1971.

[10] Claus-Peter Schnorr. The process complexity and effective random tests. In *STOC*, pages 168–176, 1972.

[11] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2, 42:230–265, 1936.

[12] Marcelo Viana. Stochastic dynamics of deterministic systems. *Lecture Notes XXI Braz. Math. Colloq. IMPA Rio de Janeiro*, 1997.

[13] J. Ville. *Etude Critique de la Notion de Collectif*. Gauthier-Villars, Paris, 1939.

[14] V'yugin V.V. Effective convergence in probability and an ergodic theorem for individual random sequences. *SIAM Theory of Probability and Its Applications*, 42(1):39–50, 1997.

[15] Klaus Weihrauch. Computability on computable metric spaces. *Theoretical Computer Science*, 113:191–210, 1993. Fundamental Study.

[16] Lai-Sang Young. What are SRB measures, and which dynamical systems have them? *J. Stat. Phys.*, 108:733–754, 2002.

[17] A.K. Zvonkin and L.A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematics Surveys*, 256:83–124, 1970.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 481–492
www.stacs-conf.org

THE DYNAMIC COMPLEXITY OF FORMAL LANGUAGES

WOUTER GELADE [1] AND MARCEL MARQUARDT [2] AND THOMAS SCHWENTICK [2]

[1] Hasselt University and Transnational University of Limburg, School for Information Technology

[2] Technische Universität Dortmund

Abstract. The paper investigates the power of the dynamic complexity classes DynFO, DynQF and DynPROP over string languages. The latter two classes contain problems that can be maintained using quantifier-free first-order updates, with and without auxiliary functions, respectively. It is shown that the languages maintainable in DynPROP exactly are the regular languages, even when allowing arbitrary precomputation. This enables lower bounds for DynPROP and separates DynPROP from DynQF and DynFO. Further, it is shown that any context-free language can be maintained in DynFO and a number of specific context-free languages, for example all Dyck-languages, are maintainable in DynQF. Furthermore, the dynamic complexity of regular tree languages is investigated and some results concerning arbitrary structures are obtained: there exist first-order definable properties which are not maintainable in DynPROP. On the other hand any existential first-order property can be maintained in DynQF when allowing precomputation.

1. Introduction

Traditional complexity theory asks for the necessary effort to decide whether a given input has a certain property, more precisely, whether a given string is in a certain language. In contrast, *dynamic complexity* asks for the effort to maintain sufficient knowledge to be able to decide whether the input object has the property *after a series of small changes of the object*. The complexity theoretic investigation of the dynamic complexity of algorithmic problems was initiated by Patnaik and Immerman [18]. They defined the class DynFO of dynamic problems where small changes in the input can be mastered by formulas of (first-order) predicate logic (or, equivalently, poly-size circuits of bounded depth, see [8]). More precisely, the dynamic program makes use of an auxiliary data structure and after each update (say, insertion or deletion) the auxiliary data structure can be adapted by a first-order formula.

Among others they showed that the dynamic complexity of the following problems is in DynFO: Reachability in undirected graphs, minimum spanning forests, multiplication, regular languages, the Dyck languages D_n. Subsequent work has yielded more problems in DynFO [8] some of which are LOGCFL-complete [20] and even PTIME-complete [17, 18] (even though the latter are highly artificial). Other work also considered stronger classes (like Hesse's result that Reachability in arbitrary directed graphs is in DynTC^0 [13]), studied

Key words and phrases: Dynamic complexity, Regular languages, Context-free languages, DynFO.
Wouter Gelade is a Research Assistant of the Fund for Scientific Research - Flanders (Belgium) .

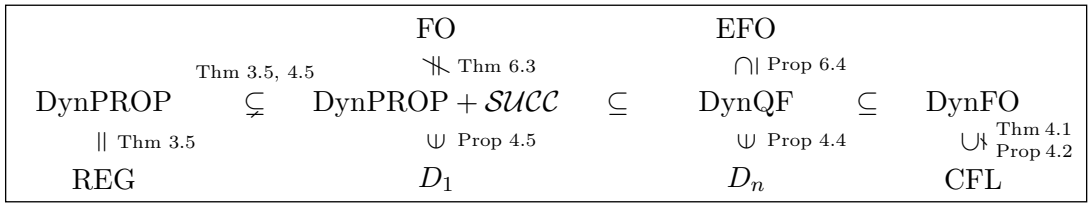


Figure 1: An overview of the main results in this paper.[1]

notions of completeness for dynamic problems [15], and elaborated on the handling of precomputations [20].

The choice of first-order logic as update language in [18] was presumably triggered by the hope that, in the light of lower bounds for AC^0, it would be possible to prove that certain problems do *not* have DynFO dynamic complexity. As it is easy to show that every DynFO problem is in PTIME, a non-trivial lower bound result would have to show that the dynamic complexity of some PTIME problem is not in DynFO. However, so far there are no results of this kind.

The inability to prove lower bounds has naturally led to the consideration of subclasses of DynFO. Hesse studied problems with quantifier-free update formulas, yielding DynPROP if the maintained data structure is purely relational and DynQF if functions are allowed as well [12, 14]. As further refinements the subclasses DynOR and DynProjections were studied. In [12] separation results for subclasses of DynPROP were shown and the separation between DynPROP and DynP was stated as an open problem.

The framework of [18] allows more general update operations and some of the results we mention depend on the actual choice of operations. Nevertheless, most research has concentrated on insertions and deletions as the only available operations. Furthermore, most work considered underlying structures of the following three kinds.

Graphs: Here, edges can be inserted or deleted. One of the main open questions is whether Reachability (aka transitive closure) can be maintained in DynFO for directed, possibly cyclic graphs.

Strings: Here, letters can be inserted or deleted. As mentioned above, [18] showed that regular languages and Dyck languages can be maintained in DynFO. Later on, Hesse proved that the dynamic complexity of regular languages is actually in DynQF [14].

Databases: The dynamic complexity of database properties were studied in the slightly different framework of First-Order Incremental Evaluation Systems (FOIES) [7]. Many interesting results were shown including a separation between deterministic and nondeterministic systems [5] and inexpressibility results for auxiliary relations of small arity [4, 6]. Nevertheless, general lower bounds have not been shown yet.

Continuing the above lines of research, this paper studies the dynamic complexity of formal languages with a particular focus on dynamic classes between DynPROP and DynQF. Our main contributions are as follows (see also Figure 1):

- We give an exact characterization of the dynamic complexity of regular languages: a language can be maintained in DynPROP if and only if it is regular. This also holds in the presence of arbitrary precomputed (aka built-in) relations. (Section 3)

[1] In this figure the dynamic complexity classes are allowed to operate with precomputation. Some of the results also hold without precomputation, for example all results concerning formal languages.

- We provide (presumably) better upper bounds for context-free languages: every context-free language can be maintained in DynFO, Dyck languages even in DynQF, Dyck languages with one kind of brackets in a slight extension of DynPROP, where built-in successor and predecessor functions can be used. (Section 4)
- As an immediate consequence, we get a separation between DynPROP and DynQF, thereby also separating DynPROP from DynFO and DynP.
- We investigate a slightly different semantics for dynamic string languages, and we show that regular tree languages can be maintained in DynPROP, when allowing precomputation and the use of built-in functions. (Section 5).
- Further, we study general structures, and show that (bounded-depth) alternating reachability is not maintainable in DynPROP. From this we can conclude that not all first-order definable properties are maintainable in DynPROP. On the other hand, we prove that all existential first-order definable properties are maintainable in DynQF when allowing precomputation. (Section 6)

Related work. We already discussed most of the related work above. A related research area is the study of incremental computation and the complexity of problems in the cell probe model. Here, the focus is not on structural (parallel) complexity of updates but rather on (sequential) update time [16, 17]. In particular, [9, 10] give efficient incremental algorithms and analyse the complexity of formal language classes based on completely different ideas.

Another area related to dynamic formal languages is the incremental maintenance of schema information (aka regular tree languages) [1, 2] and XPath query evaluation [3] in XML documents. There, the interest is mainly in fast algorithms, less in structural dynamic complexity. Nevertheless techniques of dynamic algorithms on string languages also find applications in these settings.

Due to lack of space all proofs are omitted, except for some proof sketches. They are available in the full version of the paper [11].

2. Definitions

Let $\Sigma = \{\sigma_1, ..., \sigma_k\}$ be a fixed alphabet. We represent words over Σ encoded by *word structures*, i.e., logical structures W with universe $\{1 \dots, n\}$, one unary relation R_σ for each symbol $\sigma \in \Sigma$, and the canonical linear order $<$ on $\{1 \dots, n\}$. We only consider structures in which, for each $i \leq n$, there is at most one $\sigma \in \Sigma$ such that $R_\sigma(i)$ holds, but there might be none such σ. We write $W(i) = \sigma$ if $R_\sigma(i)$ holds and $W(i) = \varepsilon$ if no such σ exists. We call n the *size* of W.

The word $w = \text{word}(W)$ represented by a word structure W is simply the concatenation $W(1) \circ \cdots \circ W(n)$. Notice that, due to the fact that certain elements in W might not carry a symbol, the actual length of the string can be less than n. In particular, every word w can be encoded by infinitely many different word structures. Let $[i, j]$ and $]i, j[$ denote the intervals from i to j, resp. from $i+1$ to $j-1$. For a word structure W, and positions $i \leq j$ in $[1, n]$, we write $w[i, j]$ for the (sub-)string $W(i) \circ \cdots \circ W(j)$. In particular $w[i, i-1]$ denotes the empty substring between positions i and $i-1$.

By E_n we denote the structure with universe $\{1, .., n\}$ representing the empty string ε (thus in E_n all relations R_σ are empty).

2.1. Dynamic Languages and Complexity Classes

In this section, we first define dynamic counterparts of formal languages. Informally, a dynamic language consists of all sequences of insertions and deletions of symbols that transform the empty string into a string of a particular (static) language L. Then we define dynamic programs which are intended to keep track of whether the string resulting from a sequence of updates is in L. Finally we define complexity classes of dynamic languages. Most of our definitions are inspired by [18] but, as we consider strings as opposed to arbitrary structures, we try to keep the formalism as simple as possible.

Dynamic Languages. We will associate with each string language L a dynamic language $\mathrm{Dyn}(L)$. The idea is that words can be changed by insertions and deletions of letters and $\mathrm{Dyn}(L)$ is basically the set of update sequences α which turn the empty string into a string in L.

For an alphabet Σ we define the set $\Delta := \{\mathrm{ins}_\sigma \mid \sigma \in \Sigma\} \cup \{\mathrm{reset}\}$ of *abstract updates*. A *concrete update* is a term of the form $\mathrm{ins}_\sigma(i)$ or $\mathrm{reset}(i)$, where i is a positive integer. A concrete update is *applicable* in a word structure of size n if $i \leq n$. By Δ_n we denote the set of applicable concrete updates for word structures of size n. If there is no danger of confusion we will simply write "update" for concrete or abstract updates.

The semantics of applicable updates is defined as expected: $\delta(W)$ is the structure resulting from W by

- setting $R_\sigma(i)$ to true and $R_{\sigma'}(i)$ to false, for $\sigma' \neq \sigma$, if $\delta = \mathrm{ins}_\sigma(i)$, and
- setting all $R_\sigma(i)$ to false, if $\delta = \mathrm{reset}(i)$.

For a sequence $\alpha = \delta_1 \dots \delta_k \in \Delta_n^+$ of updates we define $\alpha(W)$ as $\delta_k(\dots(\delta_1(W))\dots)$.

Definition 2.1. Let L be a language over alphabet Σ. The *dynamic language* $\mathrm{Dyn}(L)$ is the set of all (non-empty) sequences α of updates, for which there is $n > 0$ such that $\alpha \in \Delta_n^+$ and $\mathrm{word}(\alpha(E_n)) \in L$. We call L the *underlying language* of $\mathrm{Dyn}(L)$.[2]

Dynamic Programs. Informally, a dynamic program is a transition system which reads sequences of concrete updates and stores the current string and some auxiliary relations in its state. It also maintains the information whether the current string is in the (static) language under consideration.

A *program state* S is a word structure W extended by (auxiliary) relations over the universe of W. The *schema* of S is the set of names and arities of the auxiliary relations of S. We require that each program has a 0-ary relation ACC.

A *dynamic program* P over alphabet Σ and schema $\mathcal{R}$ consists of an *update function* $\phi^R_{\mathrm{op}}(y; x_1, \dots, x_k)$, for every $\mathrm{op} \in \Delta$ and $R \in \mathcal{R}$, where $k = \mathrm{arity}(R)$. A dynamic program P operates as follows. Let S be a program state with word structure W. The application of an applicable update $\delta = \mathrm{op}(i)$ on S yields the new state $S' = \delta(S)$ consisting of $W' = \delta(W)$ and new relations $R' = \{\bar{j} \mid S \models \phi^R_{\mathrm{op}}(i, \bar{j})\}$, for each $R \in \mathcal{R}$. For each $n \in \mathbb{N}$ and update sequence $\alpha = \delta_1 \dots \delta_k \in \Delta_n^+$ we define $\alpha(S)$ as $\delta_k(\dots(\delta_1(S))\dots)$. We say that a state S is *accepting* iff $S \models \mathrm{ACC}$, i.e., if the 0-ary ACC-relation contains the empty tuple.[3]

[2] There is a danger of confusion as we deal with two kinds of languages: "normal languages" consisting of "normal strings" and dynamic languages consisting of sequences of updates. We use the terms "word" and "string" only for "normal strings" and call the elements of dynamic languages "sequences".

[3] 0-ary relations can be viewed as propositional variables: either they contain the empty tuple (corresponding to TRUE) or not.

We say that a dynamic program P *recognizes* the dynamic language $\mathrm{Dyn}(L)$ if for all $n \in \mathbb{N}$ and all $\alpha \in \Delta_n^+$ it holds that $\alpha(E_n{}')$ is accepting iff $\mathrm{word}(\alpha(E_n)) \in L$, where $E_n{}'$ denotes the state with word structure E_n and otherwise empty relations.

Dynamic Complexity Classes. DynFO is the class of all dynamic languages that are recognized by dynamic programs whose update functions are definable by first-order formulas. DynPROP is the subclass of DynFO where all these formulas are quantifier free.

2.2. Extended Dynamic Programs

To gain more insight into the subtle mechanics of dynamic computations, we study two orthogonal extensions of dynamic programs: auxiliary functions and precomputations.

Extending dynamic programs with functions. A dynamic program P *with auxiliary functions* is a dynamic program over a schema $\mathcal{R}$, possibly containing function symbols, which has, for each $\sigma \in \Sigma$ and each function symbol $F \in \mathcal{R}$ an update function $\psi_\sigma^F(i; x_1, ..., x_k)$ where $k = \mathrm{arity}(F)$.

As we are mainly interested in quantifier free update functions for updating auxiliary functions we restrict ourselves to update functions defined by *update terms*, defined as:

- Every x_i is an update term.
- If $F \in \mathcal{R}$ is a function and $\bar{t}$ contains only update terms then $F(\bar{t})$ is an update term.
- If ϕ is a quantifier free formula (possibly using update terms) and t_1 and t_2 are update terms then $\mathbf{ite}(\phi, t_1, t_2)$ is an update term.

The semantics of update terms is straightforward for the first two rules. A term $\mathbf{ite}(\phi, t_1, t_2)$ takes the value of t_1 if ϕ evaluates to true and the value of t_2 otherwise.

After an update δ, the auxiliary functions in the new state are defined by the update functions in the straightforward way. Unless otherwise stated, the functions in the initial state $E_n{}'$ map every tuple to its first element.

Extending dynamic programs with precomputations. Sometimes it can be useful for a dynamic algorithm to have a precomputation which prepares some sophisticated data structures. Such precomputations can easily be incorporated into the model of dynamic programs.

In [18] the class DynFO^+ allowed polynomial time precomputations on the auxiliary relations. The structual properties of dynamic algorithms with precomputation were further studied and refined in [20]. In this paper, we do not consider different complexities of precomputations but distinguish only the cases where precomputations are allowed or not.

A *dynamic program P with precomputations* uses an additional set of *initial auxiliary relations* (and possibly *initial auxiliary functions*). For each initial auxiliary relation symbol R and each n, P has a relation R_n^{init} over $\{1, \ldots, n\}$. The semantics of dynamic programs with precomputations is adapted as follows: in the initial state $E_n{}'$ each initial auxiliary relation R is interpreted by R_n^{init}. Similarly, for initial auxiliary function symbol F and each n there is a function F_n^{init} over $\{1, \ldots, n\}$.

Initial auxiliary relations and functions are never updated, i.e., P does not have update functions for them.

The extension of dynamic programs by functions and precomputations can be combined and gives rise to different complexity classes: For $I \in \{\bot, \mathrm{Rel}, \mathrm{Fun}\}$ and $A \in \{\mathrm{Rel}, \mathrm{Fun}\}$ we denote by $\mathrm{DynC}(I, A)$ the class of dynamic languages recognized by dynamic programs

- without precomputations, if $I = \bot$,
- with initial auxiliary relations, if $I = \mathrm{Rel}$,
- with initial auxiliary relations and functions, if $I = \mathrm{Fun}$,
- with (updatable) auxiliary relations only, if $A = \mathrm{Rel}$, and
- with (updatable) auxiliary relations and functions, if $A = \mathrm{Fun}$.

Thus, we have $\mathrm{DynFO} = \mathrm{DynFO}(\bot, \mathrm{Rel})$ and $\mathrm{DynPROP} = \mathrm{DynPROP}(\bot, \mathrm{Rel})$. If the base class DynC is DynPROP or DynFO, $\mathrm{DynC}(I, A)$ is clearly monotonic with respect to the order $\bot < \mathrm{Rel} < \mathrm{Fun}$ In particular,

$$\mathrm{DynPROP}(\mathrm{Rel}, \mathrm{Rel}) \subseteq \mathrm{DynPROP}(\mathrm{Fun}, \mathrm{Rel}) \subseteq \mathrm{DynPROP}(\mathrm{Fun}, \mathrm{Fun})$$

As we are particularly interested in the class $\mathrm{DynPROP}(\bot, \mathrm{Fun})$ we denote it also more consisely by DynQF.

As auxiliary functions can be simulated by auxiliary relations if the update functions are first-order formulas we also have $\mathrm{DynFO}(\mathrm{Rel}, \mathrm{Rel}) = \mathrm{DynFO}(\mathrm{Fun}, \mathrm{Fun})$ and $\mathrm{DynFO} = \mathrm{DynFO}(\bot, \mathrm{Fun})$. Thus, in our setting there are only two classes with base class DynFO: the one with and the one without precomputations.

We will also examine the setting where we only allow a specific set of initial auxiliary (numerical) functions, namely built-in successor and predecessor functions. For each universe size n let SUCC be the function that maps every universe element to its successor (induced by the ordering) and the element n to itself, let PRE be the function mapping to predecessors and the element 1 to itself, and let MIN be the constant (i.e. nullary function) mapping to the minimal element 1 in the universe. Then $\mathrm{DynPROP}(\mathcal{SUCC}, \mathrm{Rel})$ is the class of dynamic languages recognized by dynamic programs using quantifier-free formulas with initial (precomputed) auxiliary relations, the auxiliary functions SUCC, PRE and MIN and updatable auxiliary relations.

3. Dynamic Complexity of Regular Languages

As already mentioned in the introduction, it was shown by Patnaik and Immerman [18] that every regular language can be recognized by a DynFO program. Hesse [14] showed that the full power of DynFO is actually not needed: every regular language is recognized by some DynQF program.

Our first result is a precise characterization of the dynamic languages $\mathrm{Dyn}(L)$ with an underlying regular language L: they exactly constitute the class DynPROP. Before stating the result formally and sketch its proof, we will give a small example to illustrate how regular languages can be maintained in DynPROP.

Example 3.1. Consider the regular language $(a+b)^*a(a+b)^*$ over the alphabet $\{a, b\}$. One has to maintain one binary relation $A(i, j)$ that is true iff $i < j$ and there exists $k \in]i, j[$ such that $w[k, k] = a$ and two unary relations $I(j) \equiv \exists k < j : w[k, k] = a$ and $F(i) \equiv \exists k > i : w[k, k] = a$. We note that it is important here that $A(i, j)$ refers to the interval $]i, j[$, and not $[i, j]$, in order to maintain these relations in DynPROP.

For the operation ins_a the update formulas are straightforward, therefore we will give here just the update formulas for A and ACC after the operation ins_b. Exactly the same formulas can be used for the reset operation.

$$\begin{aligned}\phi^{A}_{\mathrm{ins}_b}(y;x_1,x_2) &\equiv \big(y \notin]x_1,x_2[\wedge A(x_1,x_2)\big)\\ &\quad \vee \big(y \in]x_1,x_2[\wedge (A(x_1,y) \vee A(y,x_2))\big)\\ \phi^{\mathrm{ACC}}_{\mathrm{ins}_b}(y) &\equiv I(y) \vee F(y)\end{aligned}$$

■

Proposition 3.2. *For every regular language L, Dyn(L) $\in$ DynPROP.*

Proof. Let $A = (Q, \delta, s, F)$ be a DFA accepting L with transition function $\delta : Q \times \Sigma \to Q$. Let $\delta^* : Q \times \Sigma^* \to Q$ denote the reflexive-transitive closure of δ.

Like in the example above one has to maintain information about the open intervals $]i, j[$, namely whether the substring $w[i+1, j-1]$ brings the automaton from a state p to state q. Here, we only give the different auxiliary relations. For all states p and q we maintain

- $R_{p,q} = \{(i,j) \mid i < j \wedge \delta^*(p, w[i+1, j-1]) = q\}$
- $I_q = \{j \mid \delta^*(s, w[1, j-1]) = q\}$ and $F_p = \{i \mid \delta^*(p, w[i+1, n]) \in F\}$ ■

As a matter of fact, the converse of Proposition 3.2 is also true, thus DynPROP is the exact dynamic counterpart of the regular languages.

Proposition 3.3. *Let L = Dyn(L') be a dynamic language in DynPROP. Then L' is regular.*

Proof. The idea of the proof is as follows. We consider a dynamic program P for L and see what happens if, starting from the empty word, the positions of a word are set in a left-to-right fashion. Since the acceptance of the word by P does not depend on the sequence of updates used to produce the word, it suffices to care only about this one update sequence.

We make the following observations.

(1) After each update all tuples of positions that have not been set yet behave the same with respect to the auxiliary relations.
(2) There is only a bounded number (depending only on P, namely on the number and the maximal arity of the auxiliary relations) of possible ways these tuples behave.
(3) The change in behavior of the tuples by one update is uniquely determined by the inserted symbol.

Together these observations enable us to define a finite automaton for L'. ■

Remark 3.4. Proposition 3.3 is a powerful tool for proving lower bounds as it, of course, shows that, for every non-regular language L, Dyn(L) $\notin$ DynPROP.

The proof of Proposition 3.3 intuitively relies on the fact that all remaining string positions cannot be distinguished before they are set. Using a Ramsey argument, this idea can be generalized to the setting with precomputations, thus showing that (relational) precomputations do not increase the expressive power of DynPROP-programs. This fact and the above two propositions can then be combined into the following theorem.

Theorem 3.5. *Let L be a language. Then, the following are equivalent:*

(1) L is regular;
(2) Dyn(L) $\in$ DynPROP; and
(3) Dyn(L) $\in$ DynPROP(Rel, Rel). ■

4. Dynamic Complexity of Context-free Languages

In the previous section we have seen that the regular languages are exactly those languages that can be recognized by a DynPROP program. In this section, we will study the dynamic complexity of context-free languages. We first show that any context-free language can be maintained in DynFO. Later on, we exhibit languages that can be maintained in DynQF or a weak extension of DynPROP.

Theorem 4.1. *Let L be a context-free language. Then, Dyn(L) is in DynFO.*

Proof. Let L be a context-free language. Consider a grammar $G = (V, \Sigma, S, D)$ for L and assume w.l.o.g. that is in Chomsky normal form. For $U \in V$, and $w \in (V \cup \Sigma)^*$, we denote by $U \to^* w$ that w can be derived from U. Then, $L = \{w \mid w \in \Sigma^* \wedge S \to^* w\}$.

Our dynamic program P recognizing L will maintain for all $X, Y \in V$ the following relation:

$$R_{X,Y} = \{(i_1, i_2, j_1, j_2) \mid [j_1, j_2] \subseteq [i_1, i_2] \wedge X \to^* w[i_1, j_1 - 1]Y w[j_2 + 1, i_2]\}$$

■

However, we cannot hope for an equivalence between DynFO and the context-free languages, as for DynPROP and the regular languages before. This follows easily as opposed to the class of context-free languages, DynFO is closed under intersection and complement. Furthermore, one can show that non-contextfree languages can be maintained in DynQF and DynPROP($\mathcal{SUCC}$, Rel). This is because unary counters can be implemented easily by dynamic programs in these classes. Let EQUAL$_r$ be the language over the alphabet $\Sigma = \{a_1, \ldots, a_r\}$ containing all strings with an equal number of occurrences of each symbol a_i. Note that already EQUAL$_3$ is not context-free. Using the counters one can prove the following

Proposition 4.2.

(1) Dyn(EQUAL$_r$) $\in$ DynPROP($\mathcal{SUCC}$, Rel)
(2) Dyn(EQUAL$_r$) $\in$ DynQF ■

From Proposition 4.2 and Theorem 3.5 one can conclude the following

Corollary 4.3.

(1) DynPROP $\subsetneq$ DynPROP($\mathcal{SUCC}$, Rel)
(2) DynPROP $\subsetneq$ DynQF ■

One can also get better upper bounds for Dyck-languages, the languages of properly balanced parentheses. For a set of opening brackets $\{(_1, ..., (_n\}$ and the set of its closing brackets $\{)_1, ...,)_n\}$ the language D_n is the language produced by the context free grammar:

$$S \to SS \mid (_1 S)_1 \mid ... \mid (_n S)_n \mid \varepsilon$$

Proposition 4.4. *For every $n > 0$, $D_n \in$ DynQF.* ■

The proof of Proposition 4.4 mainly relies on the fact that each terminal symbol occurs in only one rule of the grammar and therefore corresponding positions (i.e. matching brackets) can be maintained using auxiliary functions. Together with relations similar to the proof of Theorem 4.1 these functions enable us to maintain D_n.

We expect the result to hold for a broader class of context-free languages which has yet to be pinned down exactly. It is even conceivable that all deterministic or unambiguous context-free languages are in DynQF.

It turns out that for Dyck languages with only one kind of brackets, i.e., D_1, auxiliary functions are not needed, if built-in successor and predecessor functions are given.

Proposition 4.5. *$D_1 \in DynPROP(\mathcal{SUCC}, Rel)$*

Proof. The idea of the proof uses the well known *level-trick* for the Dyck-languages (cf.[18]). All string positions of the same level are stored in a list, represented by the edge relation of a directed graph forming a cycle. This representations allows to infer whether there exists a string position of a given level and is maintainable in DynPROP($\mathcal{SUCC}$, Rel) ∎

Thus, whereas built-in relations did not increase the power of DynPROP, already the three simple functions SUCC, PRE and MIN allow the maintenance of non-regular languages.

5. Variations

Alternative Semantics. Following [18], we have introduced in Section 2 dynamic languages in which it is both allowed to insert or change labels at positions in the string and to delete elements at positions. In a universe of size n, one can thus create all strings of length smaller or equal than n.

However, one can also consider the setting in which each position in the string must at any time be assigned a symbol. Although this setting is less "dynamic", it has the advantage that a word is always associated with its canonical logical structure. This can be achieved by starting with an initial structure in which each symbol is already assigned a symbol, and subsequently only allowing labels to be changed (and not deleted).

More formally, we assign to every language L, a dynamic language Dyn-alt(L) as follows. For a distinguished *initial symbol* $a \in \Sigma$, and $n \in \mathbb{N}$, let E_n^a be the word structure in which $R_a(i)$ is true, for all i, and R_σ is empty, for all $\sigma \neq a$. Further, $\Delta_n = \{\text{ins}_\sigma \mid \sigma \in \Sigma\}$. Then, Dyn-alt($L$) $= \{(n, \delta) \mid \delta \in \Delta_n^+ \wedge \text{word}(\delta(E_n^a)) \in L\}$[4].

Proposition 5.1 shows that the situation is less appealing than in the original semantics. In particular, there are regular languages which cannot be maintained without precomputation; and with precomputation all regular, but also non-regular, languages can be maintained. Here, MIDDLE $= \{wbw' \mid |w| = |w'|\}$ is the language over the alphabet $\Sigma = \{a, b\}$ which contains all strings whose middle element is b, which is clearly not regular.

Proposition 5.1.

(1) Dyn-alt($L((aa)^*)$) $\notin$ DynPROP
(2) For any regular language L, Dyn-alt(L) $\in$ DynPROP(Rel, Rel)
(3) Dyn-alt(MIDDLE) $\in$ DynPROP(Rel, Rel) ∎

Notice that, contrary to Theorem 3.5, this proposition does not allow us to infer lower bounds for DynPROP(Rel, Rel) under the current semantics. However, if we consider the class of languages with neutral elements, this becomes possible again. We say that a language L has a *neutral element* a if for all $w, w' \in \Sigma^*$ it holds that $ww' \in L$ iff $waw' \in L$. Here, if a language has at least one neutral element we will assume that the initial symbol for its dynamic algorithm is one of these neutral elements.

[4]Notice that Dyn(L) consists only of update sequences δ, whereas Dyn-alt(L) contains tuples (n, δ). This change is necessary as the membership of a word of a language under the current semantics can depend both on the size of the initial structure n, and the update sequence δ.

Then, a straigthforward generalization of Theorem 3.5 yields the following proposition which implies, for instance, that Dyn-alt$(L) \notin$ DynPROP(Rel, Rel) for all non-regular languages L which have a neutral element.

Proposition 5.2. *Let L be a language which has a neutral element. Then, the following are equivalent:*

(1) L is regular;
(2) Dyn-alt$(L) \in$ DynPROP; and
(3) Dyn-alt$(L) \in$ DynPROP(Rel, Rel). ∎

Regular Tree Languages. We now investigate the dynamic complexity of the regular tree languages. Thereto, we first define dynamic tree language. A tree t over an alphabet Σ is encoded by a logical structure T with as universe the first n elements of the list $(1, 11, 12, 111, 112, 121, 122, \ldots)$, for some $n \in \mathbb{N}$, and consisting of (1) one unary relation R_σ, for each symbol $\sigma \in \Sigma$, (2) a constant root, denoting the element 1, and (3) binary relations L-child and R-child, containing all tuples $(u, u1)$ and $(u, u2)$, respectively.

The updates are terms $\text{ins}_\sigma(u)$ and reset(u), setting and resetting the label of node u in T, exactly as in the string case. So, the logical structure T is a fixed balanced binary tree in which the labels can change. Then, the tree t encoded by T is the largest subtree of T whose root is the element 1 and in which all nodes are labelled with an alphabet symbol. Notice that a node of T is included in t if it, and all its ancestors, carry an alphabet symbol.

Exactly as for the word languages, for a tree language L, we let Dyn(L) be the set of update sequences leading to a tree $t \in L$. A dynamic program works on a dynamic tree language exactly as it does on a dynamic language. We then obtain the following result.

Proposition 5.3. *Let L be a regular tree language. Then, Dyn$(L) \in$ DynPROP(Fun, Rel).* ∎

6. Beyond Formal Languages

The definitions given in Section 2 only concerned dynamic problems for word structures. Following [20], we now extend these definitions to arbitrary structures. Thereto, let γ be a vocabulary containing relation symbols of arbitrary arities. We assume that a structure over γ of size n has as universe $\{1, \ldots, n\}$. The empty structure over vocabulary γ of size n and only empty relations is denoted $E_n(\gamma)$.

The set of *abstract updates* $\Delta(\gamma)$ is defined as $\{\text{ins}_R, \text{del}_R \mid R \in \gamma\}$. A *concrete update* is a term of the form $\text{ins}_R(i_1, \ldots, i_k)$ or $\text{del}_R(i_1, \ldots, i_k)$, where $k = \text{arity}(R)$. A concrete update is *applicable* in a structure of size n if $i_j \leq n$, for all $j \in [1, k]$. By $\Delta_n(\gamma)$ we denote the set of applicable concrete updates for structures over γ of size n. For a sequence $\alpha = \delta_1 \ldots \delta_k \in (\Delta_n(\gamma))^+$ of updates we define $\alpha(A)$ as $\delta_k(\ldots(\delta_1(A))\ldots)$, where $\delta(A)$ is the structure obtained from A by setting $R(i_1, \ldots, i_k)$ to true if $\delta = \text{ins}_R(i_1, \ldots, i_k)$; and setting $R(i_1, \ldots, i_k)$ to false if $\delta = \text{del}_R(i_1, \ldots, i_k)$.

Definition 6.1. Let γ be a vocabulary, and F be a set of γ-structures. The *dynamic problem* Dyn(F) is the set of all pairs (n, α), with $n > 0$ and $\alpha \in (\Delta_n(\gamma))^+$ such that $\alpha(E_n(\gamma)) \in F$. We call F the *underlying static problem* of Dyn(F).

Dynamic programs operate on dynamic problems just as they do on dynamic languages.

Incomparability of FO and DynPROP. As we have seen in the previous sections, when restricted to monadic input schemas, DynPROP in a sense has the power of MSO. However, if we add one binary relation DynPROP cannot even capture first-order logic. This is also true if we allow the program to use precomputed functions from the set $\mathcal{SUCC}$.

Thereto we will consider *alternating graphs*, coded via the binary edge relation E and two unary relations A and B that form a decomposition of the universe V into the set of *existential* and *universal* nodes. Given a node $s \in V$, the set of all *reachable* nodes $\mathrm{Reach}(s)$ is defined as the smallest set satisfying (1) $s \in \mathrm{Reach}(s)$, (2) if $u \in A$ and there is an $v \in \mathrm{Reach}(s)$ such that $(u, v) \in E$, then $u \in \mathrm{Reach}(s)$ and (3) if $u \in B$ and for all nodes v such that $(u, v) \in E$ we have $v \in \mathrm{Reach}(s)$, then $v \in \mathrm{Reach}(s)$. Now we define ALT-REACH as the problem, given an alternating graph $G = (A \dot\cup B, E)$ and two nodes s and t, is $t \in \mathrm{Reach}(s)$. We note that ALT-REACH is P-complete (see for example [19]).

Proposition 6.2. *Dyn(ALT-REACH)* $\notin$ *DynPROP*$(\mathcal{SUCC}, \mathit{Rel})$ ∎

In fact from the proof of the above proposition one can conclude an even stronger statement. The graphs used in the proof are very restricted in the sense that the length of the longest path is bounded by a constant. Let $\text{ALT-REACH}_{\mathrm{depth}\leq d}$ be the alternating reachability problem on graphs of depth at most d. It is easily seen that $\text{ALT-REACH}_{\mathrm{depth}\leq d}$ is expressible by a FO-formula, so we get the following

Theorem 6.3. *There exists a problem $F \in$ FO such that Dyn$(F) \notin$ DynPROP$(\mathcal{SUCC}, \mathit{Rel})$.* ∎

On the other hand the reachability problem on acyclic deterministic directed graphs can be maintained in DynPROP (Hesse [14]) but cannot be expressed via an FO-formula (as can be easily seen by standard EF-games arguments). So these classes are incomparable.

Using functions to maintain EFO. Next we exhibit a class of properties which can be maintained in DynQF with precomputation. An *existential first-order (EFO)* sentence is a first-order sentence of the form $\exists x_1, \ldots x_k \phi(\bar{x})$, where $\phi(\bar{x})$ is a quantifier free formula.

Theorem 6.4. *For any EFO-definable problem F, Dyn$(F) \in$ DynPROP$(\mathit{Fun}, \mathit{Fun})$* ∎

The proof of this theorem relies on the fact that an EFO sentence can only assert whether a tuple of elements in the structure has certain properties, i.e. has a certain *type*. Then, using precomputed addition and subtraction functions, it is possible to count the number of tuples in a structure which have a certain type, and thus decide whether an EFO sentence is satisfied in the structure.

7. Conclusion

We have studied the dynamic complexity of formal languages and, by characterizing the languages maintainable in DynPROP as exactly the regular languages, obtained the first lower bounds for DynPROP. This yields a separation of DynPROP from DynQF and DynFO. We proved that every context-free language can be maintained in DynFO and investigated the power of functions for dynamic programs in maintaining specific context-free and non context-free languages.

As a modest extension we also proved a lower bound for DynPROP with built-in successor functions. Hence, we are now one step closer to proving lower bounds for DynFO, but, of course, a number of questions arise:

- Can the results on the Dyck languages be extended to show that an entire subclass of the context-free languages, such as the deterministic or unambiguous context-free languages, can be maintained in DynQF?
- We have seen that $D_1 \in \text{DynPROP}(\mathcal{SUCC}, \text{Rel})$. Can it be shown that $D_2 \notin \text{DynPROP}(\mathcal{SUCC}, \text{Rel})$?
- Can some of the lower bound techniques for DynPROP be extended to DynQF, in order to separate DynQF from DynFO, or at least from DynP? Is there a context-free language that is not maintainable in DynQF?

References

[1] A. Balmin, Y. Papakonstantinou, and V. Vianu. Incremental validation of XML documents. *ACM Trans. Database Syst.*, 29(4):710–751, 2004.
[2] D. Barbosa, A. O. Mendelzon, L. Libkin, L. Mignet, and M. Arenas. Efficient incremental validation of XML documents. In *ICDE*, pages 671–682, 2004.
[3] H. Björklund, W. Gelade, M. Marquardt, and W. Martens. Incremental XPath evaluation. In *ICDT*, 2009.
[4] G. Dong, L. Libkin, and L. Wong. Incremental recomputation in local languages. *Inf. Comput.*, 181(2):88–98, 2003.
[5] G. Dong and J. Su. Deterministic FOIES are strictly weaker. *Annals of Mathematics and Artificial Intelligence*, 19(1-2):127–146, 1997.
[6] G. Dong and J. Su. Arity bounds in first-order incremental evaluation and definition of polynomial time database queries. *J. Comput. Syst. Sci.*, 57(3):289–308, 1998.
[7] G. Dong, J. Su, and R. W. Topor. Nonrecursive incremental evaluation of datalog queries. *Annals of Mathematics and Artificial Intelligence*, 14(2-4):187–223, 1995.
[8] K. Etessami. Dynamic tree isomorphism via first-order updates to a relational database. In *Proceedings of PODS '98*, pages 235–243, 1998.
[9] G. S. Frandsen, T. Husfeldt, P. B. Miltersen, T. Rauhe, and S. Skyum. Dynamic algorithms for the Dyck languages. In *WADS*, pages 98–108, 1995.
[10] G. S. Frandsen, P. B. Miltersen, and S. Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997.
[11] W. Gelade, M. Marquardt, and T. Schwentick. The dynamic complexity of formal languages. Available from *arXiv:0812.1915 [cs.CC]*.
[12] W. Hesse. Conditional and unconditional separations of dynamic complexity classes, 2003. Unpublished manuscript, available from http://people.clarkson.edu/~whesse/ (seen Dec, 9, 2008).
[13] W. Hesse. The dynamic complexity of transitive closure is in DynTC^0. *Theor. Comput. Sci.*, 3(296):473–485, 2003.
[14] W. Hesse. *Dynamic Computational Complexity*. PhD thesis, University of Massachusetts Amherst, 2003.
[15] W. Hesse and N. Immerman. Complete problems for dynamic complexity classes. In *LICS*, pages 313–324, 2002.
[16] P. B. Miltersen. Cell probe complexity - a survey. In *FSTTCS*, 1999.
[17] P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia. Complexity models for incremental computation. *Theor. Comput. Sci.*, 130(1):203–236, 1994.
[18] S. Patnaik and N. Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997.
[19] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.
[20] V. Weber and T. Schwentick. Dynamic complexity theory revisited. *Theory Comput. Syst.*, 40(4):355–377, 2007.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 493–504
www.stacs-conf.org

A COMPLEXITY DICHOTOMY FOR PARTITION FUNCTIONS WITH MIXED SIGNS

LESLIE ANN GOLDBERG [1] AND MARTIN GROHE [2] AND MARK JERRUM [3] AND MARC THURLEY [2]

[1] Department of Computer Science, University of Liverpool, Liverpool L69 3BX, UK

[2] Institut für Informatik, Humboldt-Universität zu Berlin, 10099 Berlin, Germany

[3] School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London E1 4NS, UK

Abstract. *Partition functions*, also known as *homomorphism functions*, form a rich family of graph invariants that contain combinatorial invariants such as the number of k-colourings or the number of independent sets of a graph and also the partition functions of certain "spin glass" models of statistical physics such as the Ising model.

Building on earlier work by Dyer and Greenhill [7] and Bulatov and Grohe [6], we completely classify the computational complexity of partition functions. Our main result is a dichotomy theorem stating that every partition function is either computable in polynomial time or #P-complete. Partition functions are described by symmetric matrices with real entries, and we prove that it is decidable in polynomial time in terms of the matrix whether a given partition function is in polynomial time or #P-complete.

While in general it is very complicated to give an explicit algebraic or combinatorial description of the tractable cases, for partition functions described by a Hadamard matrices — these turn out to be central in our proofs — we obtain a simple algebraic tractability criterion, which says that the tractable cases are those "representable" by a quadratic polynomial over the field $\mathbb{F}_2$.

1. Introduction

We study the complexity of a family of graph invariants known as *partition functions* or *homomorphism functions* (see, for example, [10, 17, 18]). Many natural graph invariants can be expressed as homomorphism functions, among them the number of k-colourings, the number of independent sets, and the number of nowhere-zero k-flows of a graph. The functions also appear as the partition functions of certain "spin-glass" models of statistical physics such as the Ising model or the q-state Potts model.

Key words and phrases: Computational complexity.

Research of Leslie Goldberg and Mark Jerrum was partly funded by the EPSRC grant "The complexity of counting in constraint satisfaction problems".

Marc Thurley was supported by the Deutsche Forschungsgemeinschaft within the research training group 'Methods for Discrete Structures' (GRK 1408).

Let $A \in \mathbb{R}^{m\times m}$ be a symmetric real matrix with entries $A_{i,j}$. The *partition function* Z_A associates with every graph $G = (V, E)$ the real number

$$Z_A(G) = \sum_{\xi:V\to[m]} \prod_{\{u,v\}\in E} A_{\xi(u),\xi(v)}.$$

We refer to the row and column indices of the matrix, which are elements of $[m] := \{1, \dots, m\}$, as *spins*. We use the term *configuration* to refer to a mapping $\xi : V \to [m]$ assigning a spin to each vertex of the graph. To avoid difficulties with models of real number computation, throughtout this paper we restrict our attention to algebraic numbers. Let $\mathbb{R}_{\mathbb{A}}$ denote the set of algebraic real numbers.[1]

Our main result is a dichotomy theorem stating that for every symmetric matrix $A \in \mathbb{R}_{\mathbb{A}}^{m\times m}$ the partition function Z_A is either computable in polynomial time or #P-hard. This extends earlier results by Dyer and Greenhill [7], who proved the dichotomy for 0-1-matrices, and Bulatov and Grohe [6], who proved it for nonnegative matrices. Therefore, in this paper we are mainly interested in matrices with negative entries.

Examples

In the following, let $G = (V, E)$ be a graph with N vertices. Consider the matrices

$$S = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad C_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

It is not hard to see that $Z_S(G)$ is the number of independent sets of a graph G and $Z_{C_3}(G)$ is the number of 3-colourings of G. More generally, if A is the adjacency matrix of a graph H then $Z_A(G)$ is the number of homomorphisms from G to H. Here we allow H to have loops and parallel edges; the entry $A_{i,j}$ in the adjacency matrix is the number of edges from vertex i to vertex j.

Let us turn to matrices with negative entries. Consider

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{1.1}$$

Then $\frac{1}{2}Z_{H_2}(G)+2^{N-1}$ is the number of induced subgraphs of G with an even number of edges. Hence up to a simple transformation, Z_{H_2} counts induced subgraphs with an even number of edges. To see this, observe that for every configuration $\xi : V \to [2]$ the term $\prod_{\{u,v\}\in E} A_{\xi(u),\xi(v)}$ is 1 if the subgraph of G induced by $\xi^{-1}(2)$ has an even number of edges and -1 otherwise. Note that H_2 is the simplest nontrivial Hadamard matrix. Hadamard matrices will play a central role in this paper. Another simple example is the matrix

$$U = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

It is a nice exercise to verify that for connected G the number $Z_U(G)$ is 2^N if G is Eulerian and 0 otherwise.

A less obvious example of a counting function that can be expressed in terms of a partition function is the number of nowhere-zero k-flows of a graph. It can be shown that the number of

[1]There is a problem with the treatment of real numbers in [6], but all results stated in [6] are valid for algebraic real numbers. We use a standard representation of algebraic numbers by polynomials and standard Turing machines as our underlying model of computation.

nowhere-zero k-flows of a graph G with N vertices is $k^{-N} \cdot Z_{F_k}(G)$, where F_k is the $k \times k$ matrix with $(k-1)$s on the diagonal and -1s everywhere else. This is a special case of a more general connection between partition functions for matrices A with diagonal entries d and off diagonal entries c and certain values of the Tutte polynomial. This well-known connection can be derived by establishing certain contraction-deletion identities for the partition functions. For example, it follows from [20, Equations (3.5.4)] and [19, Equation (2.26) and (2.9)]

Complexity

Like the complexity of graph polynomials [2, 12, 14, 16] and constraint satisfaction problems [1, 3, 4, 5, 8, 11, 13], which are both closely related to our partition functions, the complexity of partition functions has already received quite a bit of a attention. Dyer and Greenhill [7] studied the complexity of counting homomorphisms from a given graph G to a fixed graph H without parallel edges. (Homomorphisms from G to H are also known as *H-colourings* of G.) They proved that the problem is in polynomial time if every connected component of H is either a complete graph with a loop at every vertex or a complete bipartite graph, and the problem is #P-hard otherwise. Note that, in particular, this gives a complete classification of the complexity of computing Z_A for symmetric 0-1-matrices A. Bulatov and Grohe [6] extended this to symmetric nonnegative matrices. To state the result, it is convenient to introduce the notion of a *block* of a matrix A. To define the blocks of A, it is best to view A as the adjacency matrix of a graph with weighted edges; then each non-bipartite connected component of this graph corresponds to one block and each bipartite connected component corresponds to two blocks. A formal definition will be given below. Bulatov and Grohe [6] proved that computing the function Z_A is in polynomial time if the row rank of every block of A is 1 and $\#P$-hard otherwise. The problem for matrices with negative entries was left open. In particular, Bulatov and Grohe asked for the complexity of the partition function Z_{H_2} for the matrix H_2 introduced in (1.1). Note that H_2 is a matrix with one block of row rank 2. As we shall see, Z_{H_2} is computable in polynomial time. Hence the complexity classification of Bulatov and Grohe does not extend to matrices with negative entries. Nevertheless, we obtain a dichotomy, and this is our main result.

Results and outline of the proofs

Theorem 1.1 (Dichotomy Theorem)**.** *Let $A \in \mathbb{R}_{\mathbb{A}}^{m \times m}$ be a symmetric matrix. Then the function Z_A either can be computed in polynomial time or is #P-hard.*

Furthermore, there is a polynomial time algorithm that, given the matrix A, decides whether Z_A is in polynomial time or #P-hard.

Let us call a matrix A *tractable* if Z_A can be computed in polynomial time and *hard* if computing Z_A is #P-hard. Then the Dichotomy Theorem states that every symmetric matrix with entries in $\mathbb{R}_{\mathbb{A}}$ is either tractable or hard. The classification of matrices into tractable and hard ones can be made explicit, but is very complicated and does not give any real insights. Very roughly, a matrix A is tractable if each of its blocks can be written as a tensor product of a positive matrix of row rank 1 and a tractable Hadamard matrix. Unfortunately, the real classification is not that simple, but for now let us focus on tractable Hadamard matrices. Recall that a Hadamard matrix is a square matrix H with entries from $\{-1, 1\}$ such that $H \cdot H^T$ is a diagonal matrix. Let $H \in \{-1, 1\}^{n \times n}$ be a symmetric $n \times n$ Hadamard matrix with $n = 2^k$. Let $\rho : \mathbb{F}_2^k \to [n]$ be a bijective mapping, which we call an *index mapping*. We

say that a multivariate polynomial $h(X_1, \ldots, X_k, Y_1, \ldots, Y_k)$ over $\mathbb{F}_2$ *symmetrically represents* H *with respect to* ρ if, for all $\mathbf{x} = (x_1, \ldots, x_k), \mathbf{y} = (y_1, \ldots, y_k) \in \mathbb{F}_2^k$, it holds that

$$h(x_1, \ldots, x_k, y_1, \ldots, y_k) = 1 \iff H_{\rho(\mathbf{x}),\rho(\mathbf{y})} = -1.$$

For example, the $\mathbb{F}_2$-polynomial $h_2(X_1, Y_1) = X_1 \cdot Y_1$ symmetrically represents the matrix H_2 with respect to the index mapping $\rho(x_1) = x_1 + 1$. The $\mathbb{F}_2$-polynomial $h_4(X_1, X_2, Y_1, Y_2) = X_1 \cdot Y_2 \oplus X_2 \cdot Y_1$ symmetrically represents the matrix

$$H_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

with respect to the index mapping $\rho(x_1, x_2) = 2 \cdot x_1 + x_2 + 1$. The qualifier "symmetrically" in "symmetrically represents" indicates that the same index mapping is applied to both $\mathbf{x}$ and $\mathbf{y}$. We will need to consider asymmetric representations later. Note that we can only represent a matrix $H \in \{-1, 1\}^{n \times n}$ by an $\mathbb{F}_2$-polynomial in this way if n is a power of 2. In this case, for every index mapping ρ there is a unique $\mathbb{F}_2$-polynomial symmetrically representing h with respect to ρ. We say that H has a *quadratic representation* if there is an index mapping ρ and an $\mathbb{F}_2$-polynomial h of degree at most 2 that symmetrically represents H with respect to ρ.

Theorem 1.2 (Complexity Classification for Hadamard Matrices)**.** *A symmetric Hadamard matrix H is tractable if it has a quadratic representation and hard otherwise.*

Hence, in particular, the matrices H_2 and H_4 are tractable. The tractability part of Theorem 1.2 is an easy consequence of the fact that counting the number of solutions of a quadratic equation over $\mathbb{F}_2$ (or any other finite field) is in polynomial time (see [9, 15]). The difficulty in proving the hardness part is that the degree of a polynomial representing a Hadamard matrix is not invariant under the choice of the index mapping ρ. However, for *normalised* Hadamard matrices, that is, Hadamard matrices whose first row and column consists entirely of +1s, we can show that either they are hard or they can be written as an iterated tensor product of the two simple Hadamard matrices H_2 and H_4. This gives us a canonical index mapping and hence a canonical representation by a quadratic $\mathbb{F}_2$-polynomial. Unfortunately, we could not find a direct reduction from arbitrary to normalised Hadamard matrices. To get a reduction, we first need to work with a generalisation of partition functions. If we view the matrix A defining a partition function as an edge-weighted graph, then this is the natural generalisation to graphs with edge and vertex weights. Let $A \in \mathbb{R}_{\mathbb{A}}^{m \times m}$ be a symmetric matrix and $D \in \mathbb{R}_{\mathbb{A}}^{m \times m}$ a diagonal matrix, which may be viewed as assigning the weight $D_{i,i}$ to each vertex i. We define the *partition function* $Z_{A,D}$ by

$$Z_{A,D}(G) = \sum_{\xi : V \to [m]} \prod_{\{u,v\} \in E} A_{\xi(u),\xi(v)} \cdot \prod_{v \in V} D_{\xi(v),\xi(v)},$$

for every graph $G = (V, E)$. As a matter of fact, we need a further generalisation that takes into account that vertices of even and odd degree behave differently when it comes to negative edge weights. For a symmetric matrix $A \in \mathbb{R}_{\mathbb{A}}^{m \times m}$ and two diagonal matrices $D, O \in \mathbb{R}_{\mathbb{A}}^{m \times m}$ we let

$$Z_{A,D,O}(G) = \sum_{\xi : V \to [m]} \prod_{\{u,v\} \in E} A_{\xi(u),\xi(v)} \cdot \prod_{\substack{v \in V \\ \deg(v) \text{ is even}}} D_{\xi(v),\xi(v)} \cdot \prod_{\substack{v \in V \\ \deg(v) \text{ is odd}}} O_{\xi(v),\xi(v)},$$

for every graph $G = (V, E)$. We call $Z_{A,D,O}$ the *parity-distinguishing partition function* (pdpf) defined by A, D, O. We show that the problem of computing $Z_{A,D,O}(G)$ is always either polynomial-time solvable or #P-hard, and we call a triple (A, D, O) *tractable* or *hard* accordingly. Obviously, if $D = O = I_m$ are identity matrices, then we have $Z_A = Z_{A,D} = Z_{A,D,O}$.

Returning to the proof of Theorem 1.2, we can show that, for every Hadamard matrix H, either H is hard or there is a normalised Hadamard matrix H' and diagonal matrices D', O' such that computing Z_H is polynomial time equivalent to computing $Z_{H',D',O'}$. Actually, we may assume D' to be an identity matrix and O' to be a diagonal matrix with entries $0, 1$ only. For the normalised matrix H' we have a canonical index mapping, and we can use this to represent the matrices D' and O' over $\mathbb{F}_2$. Then we obtain a tractability criterion that essentially says that (H', D', O') is tractable if the representation of H' is quadratic and that of O' is linear (remember that D' is an identity matrix, which we do not have to worry about).

For the proof of the Dichotomy Theorem 1.1, we actually need an extension of Theorem 1.2 that states a dichotomy for parity-distinguishing partition functions $Z_{A,D,O}$, where A is a "bipartisation" of a Hadamard matrix (this notion will be defined later). The proof sketched above can be generalised to give this extension. Then to prove the Dichotomy Theorem, we first reduce the problem of computing Z_A to the problem of computing Z_C for the connected components C of A. The next step is to eliminate duplicate rows and columns in the matrix, which can be done at the price of introducing vertex weights. Using the classification theorem for nonnegative matrices and some gadgetry, from there we get the desired reduction to parity-distinguishing partition functions for bipartisations of Hadamard matrices.

Let us finally mention that our proof shows that the Dichotomy Theorem not only holds for simple partition functions Z_A, but also for vertex-weighted and parity-distinguishing partition functions.

Preliminaries

Let $A \in \mathbb{R}_{\mathbb{A}}^{m \times n}$ be an $(m \times n)$-matrix. The entries of A are denoted by $A_{i,j}$. The ith row of A is denoted by $A_{i,*}$, and the jth column by $A_{*,j}$. By $\mathrm{abs}(A)$ we denote the matrix obtained from A by taking the absolute value of each entry in A.

Let I_m be the $m \times m$ identity matrix and let $I_{m;\Lambda}$ be the $m \times m$ matrix that is all zero except that $I_{j,j} = 1$ for $j \in \Lambda$.

The *Hadamard* product C of two $m \times n$ matrices A and B, written $C = A \circ B$, is the $m \times n$ component-wise product in which $C_{i,j} = A_{i,j}B_{i,j}$. $-A$ denotes the Hadamard product of A and the matrix in which every entry is -1.

We write $\langle u, v \rangle$ to denote the inner product (or dot product) of two vectors in $\mathbb{R}_{\mathbb{A}}^n$.

Recall that the *tensor product* (or *Kronecker product*) of an $r \times s$ matrix B and an $t \times u$ matrix C is an $rt \times su$ matrix $B \otimes C$. For $k \in [r]$, $i \in [t]$, $\ell \in [s]$ and $j \in [u]$, we have $(B \otimes C)_{(k-1)t+i,(\ell-1)u+j} = B_{k,\ell}C_{i,j}$. It is sometimes useful to think of the product in terms of rs "blocks" or "tiles" of size $t \times u$.

$$B \otimes C = \begin{pmatrix} B_{11}C & \dots & B_{1s}C \\ \vdots & \ddots & \vdots \\ B_{r1}C & \dots & B_{rs}C \end{pmatrix}$$

For index sets $I \subseteq [m], J \subseteq [n]$, we let $A_{I,J}$ be the $(|I| \times |J|)$-*submatrix* with entries $A_{i,j}$ for $i \in I$, $j \in J$. The matrix A is *indecomposable* if there are no index sets $I \subseteq [m], J \subseteq [n]$ such

that $(I,J) \neq (\emptyset,\emptyset)$, $(I,J) \neq ([m],[n])$ and $A_{i,j} = 0$ for all $(i,j) \in \big(([m]\setminus I)\times J\big) \cup \big(I \times ([n]\setminus J)\big)$. Note that, in particular, an indecomposable matrix has at least one nonzero entry. The *blocks* of a matrix are the maximal indecomposable submatrices. For every symmetric matrix $A \in \mathbb{R}^{n\times n}$ we can define a graph G with vertex set $[n]$ and edge set $\big\{\{i,j\} \bigm| A_{i,j} \neq 0\big\}$. We call the matrix A *bipartite* if the graph G is bipartite. We call A *connected* if the graph G is connected. The *connected components* of A are the maximal submatrices $A_{C,C}$ such that $G[C]$, the subgraph of G induced by $C \subseteq [n]$, is a connected component. If the connected component $G[C]$ is not bipartite then $A_{C,C}$ is a block of A. If the connected component $G[C]$ is bipartite and contains an edge then $A_{C,C}$ has the form $\begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}$, where B is a block of A. Furthermore, all blocks of A arise from connected components in this way.

For two Counting Problems f and g, we write $f \leq g$ if there is a polynomial time Turing reduction from f to g. If $f \leq g$ and $g \leq f$ holds, we write $f \equiv g$. For a symmetric matrix A and diagonal matrices D, O of the same size, EVAL(A,D,O) (EVAL(A,D), EVAL(A)) denotes the problem of computing $Z_{A,D,O}(G)$ ($Z_{A,D}(G)$, $Z_A(G)$, respectively) for an input graph G (which need not be a simple graph - it may have loops and/or multi-edges).

2. Hadamard matrices

The main focus of this section is to prove Theorem 2.2 below which is a strengthened version of Theorem 1.2. Suppose that H is an $n \times n$ Hadamard matrix and that Λ^R and Λ^C are subsets of $[n]$. It will be useful to work with the *bipartisation* M, Λ of H, Λ^R and Λ^C which we define as follows. Let $m = 2n$ and let M be the $m \times m$ matrix defined by the following equations for $i,j \in [n]$: $M_{i,j} = 0$, $M_{i,n+j} = H_{i,j}$, $M_{n+i,j} = H_{j,i}$, and $M_{n+i,n+j} = 0$. The matrix M can be broken into four "tiles" as follows.

$$M = \begin{pmatrix} 0 & H \\ H^T & 0 \end{pmatrix}.$$

Let $\Lambda = \Lambda^R \cup \{n + j \mid j \in \Lambda^C\}$. Note that the matrix $I_{m;\Lambda}$ can be decomposed naturally in terms of the tiles $I_{n;\Lambda^R}$ and $I_{n;\Lambda^C}$.

$$I_{m;\Lambda} = \begin{pmatrix} I_{n;\Lambda^R} & 0 \\ 0 & I_{n;\Lambda^C} \end{pmatrix}.$$

We identify a set of conditions on H, Λ^R and Λ^C that determine whether or not the problem EVAL$(M, I_m, I_{m;\Lambda})$ can be computed in polynomial time. We will see how this implies Theorem 1.2.

The Group Condition. For an $n \times n$ matrix H and a row index $l \in [n]$, let

$$G(H,l) := \{H_{i,*} \circ H_{l,*} \mid i \in [n]\} \cup \{-H_{i,*} \circ H_{l,*} \mid i \in [n]\}.$$

The *group condition for H* is:

(GC) For all $l \in [n]$, both $G(H,l) = G(H,1)$ and $G(H^T,l) = G(H^T,1)$.

The group condition gets its name from the fact that the condition implies that $G(H,l)$ is an Abelian group . As all elements of this group have order 2, the group condition gives us some information about the order of such matrices:

Lemma 2.1. *Let H be an $n \times n$ Hadamard matrix. If H satisfies* **(GC)** *then $n = 2^k$ for some integer k.*

The Representability Conditions. We describe Hadamard matrices H satisfying **(GC)** by $\mathbb{F}_2$-polynomials. By Lemma 2.1 these matrices have order $n = 2^k$. We extend our notion of "symmetric representation": Let $\rho^R : \mathbb{F}_2^k \to [n]$ and $\rho^C : \mathbb{F}_2^k \to [n]$ be index mappings (i.e. bijective mappings) and $X = (X_1, \ldots, X_k)$ and $Y = (Y_1, \ldots, Y_k)$. A polynomial $h(X, Y)$ over $\mathbb{F}_2$ *represents H with respect to ρ^R and ρ^C* if for all $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^k$ it holds that

$$h(\mathbf{x}, \mathbf{y}) = 1 \iff H_{\rho^R(\mathbf{x}),\rho^C(\mathbf{y})} = -1.$$

So a symmetric representation is just a representation with $\rho^R = \rho^C$. We say that the set Λ^R is *linear with respect to ρ^R* if there is a linear subvectorspace $L^R \subseteq \mathbb{F}_2^k$ a such that $\rho^R(L^R) = \Lambda^R$. Note that, if Λ^R is linear, then $|\Lambda^R| = 2^l$ for some $l \leq k$. We may therefore define a *coordinatisation of Λ^R (with respect to ρ^R)* as a linear map $\phi^R : \mathbb{F}_2^l \to \mathbb{F}_2^k$ such that $\phi^R(\mathbb{F}_2^l) = L^R$, that is Λ^R is just the image of the concatenated mapping $\rho^R \circ \phi^R$. We define the notion of linearity of Λ^C with respect to ρ^C and the coordinatisation of Λ^C with respect to ρ^C similarly. For a permutation $\pi \in S_k$ we use the shorthand $X_\pi \cdot Y := \bigoplus_{i=1}^k X_{\pi(i)} \cdot Y_i$.

The following conditions stipulate the representability **(R)** of H by $\mathbb{F}_2$-polynomials, the linearity **(L)** of the sets Λ^R and Λ^C, and the appropriate degree restrictions on the associated polynomials **(D)**.

(R) There are index mappings $\rho^R : \mathbb{F}_2^k \to [n]$ and $\rho^C : \mathbb{F}_2^k \to [n]$ and a permutation $\pi \in S_k$ such that (w.r.t. ρ^R and ρ^C) the matrix H is represented by a polynomial of the form

$$h(X, Y) = X_\pi \cdot Y \oplus g^R(X) \oplus g^C(Y). \tag{2.1}$$

Moreover, if Λ^R is non-empty, then $\rho^R(0) \in \Lambda^R$. Similarly, if Λ^C is non-empty, then $\rho^C(0) \in \Lambda^C$.

Finally, if H is symmetric and $\Lambda^R = \Lambda^C$, then $g^R = g^C$ and $\rho^R = \rho^C$.

(L) Λ^R and Λ^C are linear with respect to ρ^R and ρ^C respectively.

(D) Either Λ^R is empty or there is a coordinatisation ϕ^R of Λ^R w.r.t ρ^R such that the polynomial $g^R \circ \phi^R$ has degree at most 2. Similarly, either Λ^C is empty or there is a coordinatisation ϕ^C of Λ^C w.r.t ρ^C such that the polynomial $g^C \circ \phi^C$ has degree at most 2. Finally, if H is symmetric and $\Lambda^R = \Lambda^C$ is nonempty then $\phi^R = \phi^C$.

Actually, it turns out that condition (D) is invariant under the choice of the coordinatisations ϕ^R, ϕ^C. However, the conditions are not invariant under the choice of the representation ρ^R, ρ^C, and this is a major source of technical problems.

Before we can apply the conditions **(R)**, **(L)** and **(D)** we deal with one technical issue. Let H be an $n \times n$ Hadamard matrix and let $\Lambda^R, \Lambda^C \subseteq [n]$ be subsets of indices. Let M, Λ be the bipartisation of H, Λ^R and Λ^C. We say that H is *positive* for Λ^R and Λ^C if there is an entry $H_{i,j} = +1$ such that (1) $i \in \Lambda^R$ or $\Lambda^R = \emptyset$, (2) $j \in \Lambda^C$ or $\Lambda^C = \emptyset$, and (3) If H is symmetric and $\Lambda^R = \Lambda^C$ then $i = j$. Otherwise, note that $-H$ is positive for Λ^R and Λ^C. Since $Z_{M,I_m,I_{m;\Lambda}}(G) = (-1)^{|E(G)|} Z_{-M,I_m,I_{m;\Lambda}}(G)$, the problems $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ and $\mathrm{EVAL}(-M, I_m, I_{m;\Lambda})$ have equivalent complexity, so we lose no generality by restricting attention to the positive case, which is helpful for a technical reason.

Theorem 2.2. *Let H be an $n \times n$ Hadamard matrix and let $\Lambda^R, \Lambda^C \subseteq [n]$ be subsets of indices. Let M, Λ be the bipartisation of H, Λ^R and Λ^C and let $m = 2n$. If H is positive for Λ^R and Λ^C then* $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ *is polynomial-time computable if, and only if, H Λ^R and Λ^C satisfy the group condition* **(GC)** *and conditions* **(R)**, **(L)**, *and* **(D)**. *Otherwise* $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ *is #P-hard. If H is not positive for Λ^R and Λ^C then* $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ *is polynomial-time*

computable if, and only if, $-H$ Λ^R and Λ^C satisfy the group condition **(GC)** *and conditions* **(R)**, **(L)**, *and* **(D)**. *Otherwise* $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ *is* #P*-hard. There is a polynomial-time algorithm that takes input H, Λ^R and Λ^C and decides whether* $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ *is polynomial-time computable or* #P*-hard.*

The theorem is proved using a sequence of lemmas.

Lemma 2.3 (Group Condition Lemma)**.** *Let H be an $n \times n$ Hadamard matrix and let $\Lambda^R, \Lambda^C \subseteq [n]$ be subsets of indices. Let M, Λ be the bipartisation of H, Λ^R and Λ^C and let $m = 2n$. If H does not satisfy* **(GC)** *then* $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ *is* #P*-hard. There is a polynomial-time algorithm that takes determines whether H satisfies* **(GC)**.

Proof sketch. For any integer p and a symmetric non-negative matrix $C^{[p]}$, which depends upon H, the proof uses gadgetry to transform an input to $\mathrm{EVAL}(C^{[p]})$ into an input to $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$. The fact that H does not satisfy **(GC)** is used to show that, as long as p is sufficiently large with respect to M, then $C^{[p]}$ has a block of rank greater than one. By a result of Bulatov and Grohe, $\mathrm{EVAL}(C^{[p]})$ is #P-hard, so $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ is #P-hard.

Lemma 2.4 (Polynomial Representation Lemma)**.** *Let H be an $n \times n$ Hadamard matrix and $\Lambda^R, \Lambda^C \subseteq [n]$ subsets of indices. Suppose that H satisfies* **(GC)** *and that H is positive for Λ^R and Λ^C. Then the Representability Condition* **(R)** *is satisfied. There is a polynomial-time algorithm that computes the representation.*

Proof sketch. The representation is constructed inductively. First, permutations are used to transform H into a normalised matrix $\hat{H}$, that is, a Hadamard matrix $\hat{H}$ whose first row and column consist entirely of +1s, which still satisfies **(GC)**. We then show that there is a permutation of $\hat{H}$ which can be expressed as the tensor product of a simple Hadamard matrix (either H_2 or H_4) and a smaller normalised symmetric Hadamard matrix H'. By induction, we construct a representation for H' and use this to construct a representation for the normalised matrix $\hat{H}$ of the form $X_\pi \cdot Y$ for a permutation $\pi \in S_k$. We use this to construct a representation for H.

Lemma 2.5 (Linearity Lemma)**.** *Let H be an $n \times n$ Hadamard matrix and $\Lambda^R, \Lambda^C \subseteq [n]$ subsets of indices. Let M, Λ be the bipartisation of H, Λ^R and Λ^C and let $m = 2n$. Suppose that* **(GC)** *and* **(R)** *are satisfied. Then the problem* $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ *is* #P*-hard unless the Linearity condition* **(L)** *holds. There is a polynomial-time algorithm that determines whether* **(L)** *holds.*

Proof sketch. For a symmetric non-negative matrix C, which depends upon H, the proof uses gadgetry to transform an input to $\mathrm{EVAL}(C, I_m, I_{m;\Lambda})$ to an input of $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$. By **(R)**, there are bijective index mappings $\rho^R : \mathbb{F}_2^k \to [n]$ and $\rho^C : \mathbb{F}_2^k \to [n]$ and a permutation $\pi \in S_k$ such that (w.r.t. ρ^R and ρ^C) the matrix H is represented by a polynomial of the appropriate form. Let τ^R be the inverse of ρ^R and τ^C be the inverse of ρ^C. Let $L^C = \tau^C(\Lambda^C)$ and $L^R = \tau^R(\Lambda^R)$. We show that either $\mathrm{EVAL}(C, I_m, I_{m;\Lambda})$ is #P-hard or **(L)** is satisfied. In particular, the assumption that $\mathrm{EVAL}(C, I_m, I_{m;\Lambda})$ is not #P-hard means that its blocks all have rank 1 by the result of Bulatov and Grohe. We use this fact to show that L^R and L^C are linear subspaces of $\mathbb{F}_2^k$. To show that L^R is a linear space of $\mathbb{F}_2^k$, we use L^R to construct an appropriate linear subspace and compare Fourier coefficients to see that it is in fact L^R itself.

Lemma 2.6 (Degree Lemma)**.** *Let H be an $n \times n$ Hadamard matrix and $\Lambda^R, \Lambda^C \subseteq [n]$ subsets of indices. Let M, Λ be the bipartisation of H, Λ^R and Λ^C and let $m = 2n$. Suppose that* **(GC)**,**(R)** *and* **(L)** *are satisfied. Then* $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ *is* #P*-hard unless the Degree Condition* **(D)** *holds. There is a polynomial-time algorithm that determines whether* **(D)** *holds.*

Proof sketch. For any (even) integer p and a symmetric non-negative matrix $C^{[p]}$, which depends upon H, the proof uses gadgetry to transform an input to $\mathrm{EVAL}(C^{[p]})$ into an input to $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$. Using the representation of H, a coordinatisation ϕ^R with respect to Λ^R, and a coordinatisation ϕ^C with respect to Λ^C, some of the entries $C^{[p]}_{a,b}$ of the matrix $C^{[p]}$ may be expressed as sums, over elements in $\mathbb{F}_2^\ell$, for some ℓ, of appropriate powers of -1. We study properties of polynomials $g(X_1, \ldots, X_k) \in \mathbb{F}_2[X_1, \ldots, X_k]$, discovering that the number of roots of a certain polynomial $g_{\alpha,\beta,\gamma}(X_1, \ldots, X_k)$, which is derived from $g(X_1, \ldots, X_k)$, depends upon the degree of g. From this we can show that if **(D)** does not hold then there is an even p such that $\mathrm{EVAL}(C^{[p]})$ is #P-hard.

Proof of Theorem 2.2. By the equivalence of the problems $\mathrm{EVAL}(M, I_m, I_{m;\Lambda})$ and $\mathrm{EVAL}(-M, I_m, I_{m;\Lambda})$ we can assume that H is positive for Λ^R and Λ^C. The hardness part follows directly from the Lemmas above. We shall give the proof for the tractability part. Given H, Λ^R and Λ^C satisfying **(GC)**, **(R)**, **(L)** and **(D)**, we shall show how to compute $Z_{M,I_m,I_{m;\Lambda}}(G)$ for an input graph G in polynomial time.

Note first that $Z_{M,I_m,I_{m;\Lambda}}(G) = 0$ unless G is bipartite. If G has connected components $G_1, \ldots G_c$, then

$$Z_{M,I_m,I_{m;\Lambda}}(G) = \prod_{i=1}^{c} Z_{M,I_m,I_{m;\Lambda}}(G_i).$$

Therefore, it suffices to give the proof for connected bipartite graphs. Let $G = (V, E)$ be such a graph with vertex bipartition $U \dot\cup W = V$. Let $V_o \subseteq V$ be the set of odd-degree vertices in G and let $U_o = W \cap V_o$ and $W_o = W \cap V_o$ be the corresponding subsets of U and W. Let $U_e = U \setminus U_o$ and $W_e = W \setminus W_o$. We have

$$Z_{M,I_m,I_{m;\Lambda}}(G) \;=\; \sum_{\xi:V\to[m]} \prod_{\{u,w\}\in E} M_{\xi(u),\xi(w)} \prod_{v\in V_o} (I_{m;\Lambda})_{\xi(v),\xi(v)} = \sum_{\substack{\xi:V\to[m]\\ \xi(V_o)\subseteq\Lambda}} \prod_{\{u,w\}\in E} M_{\xi(u),\xi(w)}.$$

As G is bipartite and connected this sum splits into $Z_{M,I_m,I_{m;\Lambda}}(G) = Z^{\rightarrow} + Z^{\leftarrow}$ for values

$$Z^{\rightarrow} = \sum_{\substack{\xi:U\to[n]\\ \xi(U_o)\subseteq\Lambda^R}} \sum_{\substack{\zeta:W\to[n]\\ \zeta(W_o)\subseteq\Lambda^C}} \prod_{\substack{\{u,w\}\in E\\ u\in U}} H_{\xi(u),\zeta(w)} \quad \text{and} \quad Z^{\leftarrow} = \sum_{\substack{\xi:U\to[n]\\ \xi(U_o)\subseteq\Lambda^C}} \sum_{\substack{\zeta:W\to[n]\\ \zeta(W_o)\subseteq\Lambda^R}} \prod_{\substack{\{u,w\}\in E\\ u\in U}} H_{\zeta(w),\xi(u)}$$

We will show how to compute $Z^{\rightarrow}$. The computation of the value $Z^{\leftarrow}$ is similar.

Fix configurations $\xi : U \to [n]$ and $\zeta : W \to [n]$ and let ρ^R, ρ^C be the index mappings and h the $\mathbb{F}_2$-polynomial representing H as given in condition **(R)**. Let τ^R be the inverse of ρ^R and let τ^C be the inverse of ρ^C. Let $L^R = \tau^R(\Lambda^R)$ and $L^C = \tau^C(\Lambda^C)$. Then ξ and ζ induce a configuration $\varsigma : V \to \mathbb{F}_2^k$ defined by

$$\varsigma(v) := \begin{cases} \tau^R(\xi(v)) & , \text{ if } v \in U \\ \tau^C(\zeta(v)) & , \text{ if } v \in W \end{cases}$$

which implies, for all $u \in U, w \in W$ that $h(\varsigma(u), \varsigma(w)) = 1$ iff $H_{\xi(u),\zeta(w)} = -1$. Let ϕ^R and ϕ^C be coordinatisations of Λ^R and Λ^C w.r.t. ρ^R and ρ^C satisfying **(L)** and **(D)**. We can simplify

$$\begin{aligned} Z^{\rightarrow} &= \sum_{\substack{\xi:U\to[n]\\ \xi(U_o)\subseteq\Lambda^R}} \sum_{\substack{\zeta:W\to[n]\\ \zeta(W_o)\subseteq\Lambda^C}} \prod_{\substack{\{u,w\}\in E\\ u\in U}} (-1)^{h(\tau^R(\xi(u)),\tau^C(\zeta(w)))} \\ &= \sum_{\substack{\varsigma:V\to\mathbb{F}_2^k\\ \varsigma(U_o)\subseteq L^R\\ \varsigma(W_o)\subseteq L^C}} (-1)^{\bigoplus_{\{u,w\}\in E:u\in U} h(\varsigma(u),\varsigma(w))} \end{aligned}$$

Define, for $a \in \mathbb{F}_2$, sets

$$s_a := \left| \left\{ \varsigma : V \to \mathbb{F}_2^k \mid \varsigma(U_o) \subseteq L^R,\ \varsigma(W_o) \subseteq L^C,\ \bigoplus_{\substack{\{u,w\}\in E\\ u\in U}} h(\varsigma(u),\varsigma(w)) = a \right\} \right|. \tag{2.2}$$

Then $Z^{\rightarrow} = s_0 - s_1$. Therefore, it remains to show how to compute the values s_a. Define, for each $v \in V$, a tuple $X^v = (X_1^v, \ldots, X_k^v)$ and let h_G be the $\mathbb{F}_2$-polynomial

$$h_G := \bigoplus_{\substack{\{u,w\}\in E\\ u\in U}} h(X^u, X^w) = \bigoplus_{\substack{\{u,w\}\in E\\ u\in U}} (X^u)_\pi \cdot X^w \oplus \bigoplus_{u\in U_o} g^R(X^u) \oplus \bigoplus_{w\in W_o} g^C(X^w). \tag{2.3}$$

Here the second equality follows from the definition of the polynomial h given in condition **(R)** and the fact that the terms $g^R(X^u)$ and $g^C(X^w)$ in the definition of h appear exactly $\deg(u)$ and $\deg(w)$ many times in h_G. Therefore, these terms cancel for all even degree vertices.

Let $\mathrm{var}(h_G)$ denote the set of variables in h_G and for mappings $\chi : \mathrm{var}(h_G) \to \mathbb{F}_2$ we use the expression $\chi(X^v) := (\chi(X_1^v), \ldots, \chi(X_k^v))$ as a shorthand and define the $\mathbb{F}_2$-sum $h_G(\chi) := \bigoplus_{\{u,w\}\in E:u\in U} h(\chi(X^u), \chi(X^w))$. We find that s_a can be expressed by

$$s_a = \left| \left\{ \chi : \mathrm{var}(h_G) \to \mathbb{F}_2 \mid \begin{array}{ll} \chi(X^u) \in L^R & \text{for all } u \in U_o, \\ \chi(X^w) \in L^C & \text{for all } w \in W_o, \end{array} h(\chi) = a \right\} \right| \tag{2.4}$$

By equation (2.4) we are interested only in those assignments χ of the variables of h_G which satisfy $\chi(X^u) \in L^R$ and $\chi(X^w) \in L^C$ for all $u \in U_o$ and $w \in W_o$. With $|\Lambda^R| = 2^{\ell^R}$ and $|\Lambda^C| = 2^{\ell^C}$ for some appropriate ℓ^R, ℓ^C, we introduce variable vectors $Y^u = (Y_1^u, \ldots, Y_{\ell^R}^u)$ and $Z^w = (Z_1^w, \ldots, Z_{\ell^C}^w)$ for all $u \in U_o$ and $w \in W_o$. If $u \in U_o$ or $w \in W_o$ then we can express the term $(X^u)_\pi \cdot X^w$ in h_G in terms of these new variables. In particular, let

$$\begin{aligned} h''_G &= \bigoplus_{\substack{\{u,w\}\in E\\ u\in U_o, w\in W_o}} (\phi^R(Y^u))_\pi \cdot \phi^C(Z^w) \oplus \bigoplus_{\substack{\{u,w\}\in E\\ u\in U_e, w\in W_e}} (X^u)_\pi \cdot X^w \\ &\quad \oplus \bigoplus_{\substack{\{u,w\}\in E\\ u\in U_e, w\in W_o}} (X^u)_\pi \cdot \phi^C(Z^w) \oplus \bigoplus_{\substack{\{u,w\}\in E\\ u\in U_o, w\in W_e}} (\phi^R(Y^u))_\pi \cdot X^w. \end{aligned}$$

Let

$$h'_G = h''_G \oplus \bigoplus_{u\in U_o} g^R(\phi^R(Y^u)) \oplus \bigoplus_{w\in W_o} g^C(\phi^C(Z^w)) \tag{2.5}$$

We therefore have

$$s_a = \left| \left\{ \chi : \mathrm{var}(h'_G) \to \mathbb{F}_2 \mid h'_G(\chi) = a \right\} \right|. \tag{2.6}$$

By condition **(D)**, the polynomials $g^R \circ \phi^R$ and $g^C \circ \phi^C$ are of degree at most 2 and therefore h'_G is a polynomial of degree at most 2. Furthermore, we have expressed s_a as the number of solutions to a polynomial equation over $\mathbb{F}_2$. Therefore, the proof now follows by the following well-known fact.

Fact 2.7. *The number of solutions to polynomial equations of degree at most* 2 *over* $\mathbb{F}_2$ *can be computed in polynomial time.*

This is a direct consequence of Theorems 6.30 and 6.32 in [15] (see also [9]). ∎

3. The General Case

In this section we will prove Theorem 1.1. Before we can give the proof some further results have to be derived, which then enable us to extend Theorems 1.2 and 2.2. It will be convenient to focus on connected components. This is expressed by the following Lemma.

Lemma 3.1. *Let A be a symmetric matrix with entries in $\mathbb{R}_{\mathbb{A}}$ and let $A_1, \ldots, A_c$ denote its components. Then the following holds*

(1) *If* EVAL(A_i) *is* #P*-hard for some $i \in [c]$ then* EVAL(A) *is* #P*-hard.*
(2) *If* EVAL(A_i) *is* PTIME *computable for all $i \in [c]$ then* EVAL(A) *is* PTIME *computable.*

Recall that for each connected symmetric matrix A there is a block B such that either $A = B$ or, up to permutation of the rows and columns, $A = \begin{pmatrix} 0 & B \\ B^T & 0 \end{pmatrix}$. We call B the block *underlying* A. For such connected A we furthermore see that the evaluation problem is either #P-hard or we can reduce it to the evaluation problem on bipartisations of Hadamard matrices.

Lemma 3.2. *Suppose that A is a symmetric connected matrix.*
Then either EVAL(A) *is* #P*-hard or the following holds.*

(1) *If A is not bipartite there is a symmetric $r \times r$ Hadamard matrix H and a set $\Lambda^R \subseteq [r]$ such that*
$$\mathrm{EVAL}(A) \equiv \mathrm{EVAL}(H, I_r, I_{r;\Lambda^R}).$$
(2) *If A is bipartite then there is an $r \times r$ Hadamard matrix H, sets $\Lambda^R, \Lambda^C \subseteq [r]$ and a bipartisation M, Λ of H, Λ^R and Λ^C such that*
$$\mathrm{EVAL}(A) \equiv \mathrm{EVAL}(M, I_{2r}, I_{2r;\Lambda}).$$

Furthermore it can be decided in time polynomial in the size of A which of the three alternatives (#P*-hardness, (1), or (2)) holds.*

We are now able to prove the main Theorem.

Proof of Theorem 1.1. Given a symmetric matrix $A \in \mathbb{R}_{\mathbb{A}}^{m \times m}$. By Lemma 3.1 we may assume that the matrix A is connected. By Lemma 3.2, Theorem 2.2 the problem EVAL(A) is either polynomial time computable or #P-hard. The existence of a polynomial time algorithm for deciding which of the two possibilities holds, given a matrix A, follows directly by these results. ∎

References

[1] L. Barto, M. Kozik, and T. Niven. Graphs, polymorphisms and the complexity of homomorphism problems. In *Proceedings of the 40th ACM Symposium on Theory of Computing*, 2008. To appear.

[2] Markus Bläser and Holger Dell. Complexity of the cover polynomial. In L. Arge, Ch. Cachin, T. Jurdzinski, and A. Tarlecki, editors, *of the 34th International Colloquium on Automata, Languages and Programming*, volume 4596 of *Lecture Notes in Computer Science*, pages 801–812. Springer Verlag, 2007.

[3] A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53:66–120, 2006.

[4] A. Bulatov. The complexity of the counting constraint satisfaction problem. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science. Springer Verlag, 2008. To appear.

[5] A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 562–571, 2003.

[6] A. Bulatov and M. Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348:148–186, 2005.

[7] M. Dyer and C. Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 17(3–4):260–289, 2000.

[8] M.E. Dyer, L.A. Goldberg, and M. Paterson. On counting homomorphisms to directed acyclic graphs. *Journal of the ACM*, 54(6), 2007.

[9] Andrzej Ehrenfeucht and Marek Karpinski. The computational complexity of (*xor, and*)-counting problems. Technical Report 8543-CS, 1990. Available at `http://citeseer.ist.psu.edu/ehrenfeucht90computational.html`.

[10] M. Freedman, L. Lovász, and A. Schrijver. Reflection positivity, rank connectivity, and homomorphism of graphs. *Journal of the American Mathematical Society*, 20:37–51, 2007.

[11] L. A. Goldberg, S. Kelk, and M. Paterson. The complexity of choosing an H-colouring (nearly) uniformly at random. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 53–62, 2002.

[12] L.A. Goldberg and M. Jerrum. Inapproximability of the tutte polynomial. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 459–468, 2007.

[13] P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990.

[14] F. Jaeger, D. L. Vertigan, and D. J. A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society*, 108:35–53, 1990.

[15] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2nd edition, 1997.

[16] M. Lotz and J.A. Makowsky. On the algebraic complexity of some families of coloured tutte polynomials. *Advances in Applied Mathematics*, 32:327–349, 2004.

[17] L. Lovász. The rank of connection matrices and the dimension of graph algebras. *European Journal of Combinatorics*, 27:962–970, 2006.

[18] L. Lovász and A. Schrijver. Graph parameters and semigroup functions. *European Journal of Combinatorics*. To appear.

[19] Alan Sokal. The multivariate Tutte polynomial. In *Surveys in Combinatorics*. Cambridge University Press, 2005.

[20] D. J. A. Welsh. *Complexity: Knots, Colourings and Counting*, volume 186 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1993.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 505–516
www.stacs-conf.org

ASYMPTOTICALLY OPTIMAL LOWER BOUNDS ON THE NIH-MULTI-PARTY INFORMATION COMPLEXITY OF THE AND-FUNCTION AND DISJOINTNESS

ANDRÉ GRONEMEIER [1]

[1] Technische Universität Dortmund, Lehrstuhl Informatik 2, 44227 Dortmund, Germany
E-mail address: andre.gronemeier@cs.uni-dortmund.de

Abstract. Here we prove an asymptotically optimal lower bound on the information complexity of the k-party disjointness function with the unique intersection promise, an important special case of the well known disjointness problem, and the AND_k-function in the number in the hand model. Our $\Omega(n/k)$ bound for disjointness improves on an earlier $\Omega(n/(k \log k))$ bound by Chakrabarti *et al.* (2003), who obtained an asymptotically tight lower bound for one-way protocols, but failed to do so for the general case. Our result eliminates both the gap between the upper and the lower bound for unrestricted protocols and the gap between the lower bounds for one-way protocols and unrestricted protocols.

1. Introduction

Primarily, communication complexity, introduced by Yao [10], deals with the amount of communication that is needed in distributed computation, but apart from distributed computation, nowadays communication complexity has found applications in virtually all fields of complexity theory. The book by Kushilevitz and Nisan [9] gives a comprehensive introduction to communication complexity and its applications.

Suppose that k players, each of them knowing exactly one argument of a function $f(x_1, \ldots, x_k)$ with k arguments, want to evaluate the function for the input that is distributed among them. Clearly, to succeed at this task the players need to communicate. Here we consider the case that the players communicate by writing to a blackboard that is shared by all players. The rules that determine who writes which message to the blackboard are usually called a *protocol*. The protocol terminates if the value of the function can be inferred from the contents of the blackboard, the so-called *transcript* of the protocol. Then the communication complexity of the function is the minimum number of bits that the players need to write to the blackboard in the worst case to jointly compute the result. This setting is usually called the *number in the hand model* since each part of the input is exclusively known to a single player who figuratively hides the input in his hand. In the randomized version of this model each player has access to a private source of unbiased independent random bits and his actions may depend on his input and his random bits. For a *randomized ε-error protocol* the output of the protocol may be different from the value

Key words and phrases: computational complexity, communication complexity.

of the function f with probability at most ε. The ε-error randomized communication complexity of a function is defined in the obvious way. A formal definition of k-party protocols can be found in [9]. Note that there are also other models of multi-party communication, but these models are not the topic of this paper.

In recent publications [5, 2, 3, 4] lower bounds on the communication complexity of functions have been obtained by using information theoretical methods. In this context communication complexity is supplemented by an information theoretical counterpart, the information complexity of a function. Roughly, the information complexity of a function f is the minimal amount of information that the transcript of a protocol for f must reveal about the input. Besides being a lower bound for the communication complexity, information complexity has additional nice properties with respect to so-called direct sum problems.

1.1. Our Result

In this paper we will prove an asymptotically optimal lower bound on the communication complexity of the multi-party set disjointness problem with the unique intersection promise.

Definition 1.1. In the k-party set disjointness problem each of the players is given the characteristic vector of a subset of an n-element set. It is promised that the subsets are either pairwise disjoint or that there is a single element that is contained in all subsets and that the subsets are disjoint otherwise. The players have to distinguish these two cases, the output of a protocol for set disjointness should be 0 in the first case and 1 in the second case. If the promise is broken, then the players may give an arbitrary answer.

Here we will prove the following result about the randomized communication complexity of the multi-party set disjointness problem in the number in the hand model.

Theorem 1.2. *For every sufficiently small constant $\varepsilon > 0$ the randomized ε-error communication complexity of the k-party set disjointness problem with the unique intersection promise is bounded from below by $\Omega(n/k)$.*

By the upper bound shown in [4] this result is asymptotically optimal with respect to the number of players k and the size of the inputs n. An important application of this problem is the proof of a lower bound for the memory requirements of certain data stream algorithms [1]. Our improvement of the lower bound for disjointness does not have a significant impact on this application. But we think that the disjointness problem is interesting and important on its own since it is a well-known basic problem in communication complexity theory [1, 3, 4, 9]. Up to now the best known lower bound was $\Omega(n/(k \log k))$ by Chakrabarti, Khot, and Sun [4], who also proved an asymptotically optimal lower bound for one-way protocols. This result left a gap both between the upper and the lower bound and between the lower bounds for one-way protocols and unrestricted protocols. Our result closes these gaps.

Like the earlier results, our lower bound is based on an information theoretical approach. The main ingredient of this approach is a lower bound on the information complexity of the AND_k-function, the Boolean conjunction of k bits. Since Theorem 1.2 will be a simple corollary of this result, and more importantly, since AND_k is a basic building block of any computation, the lower bound on the information complexity of AND_k is the main result of this paper. We postpone the precise statement of this result to Theorem 3.2 in

Section 3 because some preparing definitions are needed beforehand. But we stress here that our result also closes the gap between the upper and lower bound on the conditional information complexity of AND_k for unrestricted protocols and the gap between the lower bounds on the information complexity of AND_k for one-way protocols and unrestricted protocols that was left open in [4].

1.2. Related Work

The general disjointness problem without the unique intersection promise has a long history in communication complexity theory. Here we focus only on recent results for the multi-party set disjointness problem with the unique intersection promise, and especially on lower bounds that rely on information complexity arguments. For older results we refer the reader to the book by Kushilevitz and Nisan [9] and the references therein.

Alon, Matias, and Szegedy [1] proved an $\Omega(n/k^4)$ lower bound for multi-party set disjointness and applied this bound to prove lower bounds for the memory requirements of data stream algorithms. Bar-Yossef, Jayram, Kumar, and Sivakumar [3] improved this to a lower bound of $\Omega(n/k^2)$. They introduced the direct sum approach on which later results, including our result, are based and proved that the information complexity of AND_k is bounded from below by $\Omega(1/k^2)$. Chakrabarti, Khot, and Sun [4] improved the lower bound for the information complexity of AND_k to $\Omega(1/(k \log k))$ and thereby improved the lower bound for multi-party set disjointness to $\Omega(n/(k \log k))$. They also proved an asymptotically optimal lower bound for one-way protocols, a restricted model in which the players communicate in a predetermined order. Our result improves on these results, but furthermore we think that our proof technique is a useful contribution to the framework for which Bar-Yossef *et al.* [3] coined the term "information statistics". Bar-Yossef *et al.* use this term for the combination of information theory and other statistical metrics on probability spaces. We use the direct sum approach from [3], but instead of the Hellinger distance that is used in [3] we use the Kullback Leibler distance. Since the Kullback Leibler distance is closely related to mutual information, we do not loose precision in the transition from information theory to statistical distance measures. By this, we are able to prove sharper bounds. Like Chakrabarti *et al.* [4], we take a closer look at the analytical properties of the functions that are involved. Our improvements on this result are also due to the fact that our Kullback Leibler distance based arguments are very close to the information theory domain.

2. Preliminaries

2.1. Notation

We use lower case letters for constants and variables and upper case letters for random variables. If the random variables X and Y have the same distribution, we briefly write $X \sim Y$. For vector-valued variables we use a boldface font. For example, $\mathbf{X} = (X_1, \ldots, X_k)$ is a random vector whose components are the random variables X_i for $i = 1, \ldots, k$. In this case let $\mathbf{X}_{-i} = (X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_k)$ denote the vector $\mathbf{X}$ without the ith component. A boldface zero $\mathbf{0}$ and boldface one $\mathbf{1}$ denote the all-zero vector and all-one vector of appropriate size, respectively. Thus $\mathbf{X}_{-i} = \mathbf{0}$ says that $X_j = 0$ for all $j \in \{1, \ldots, k\} - \{i\}$. For sums like $\sum_{i=0}^{n} a_i$ we sometimes do not explicitly specify the bounds of summation and

just write $\sum_i a_i$. In this case the sum is taken over the set of all values of i for which a_i is meaningful. This set must be derived from context. For example, the sum $\sum_v f(\Pr\{X=v\})$ should be taken over all values v in the range of X. All logarithms, denoted by log, are with respect to base 2.

2.2. Information Theory

Here we can merely define our notation for the basic quantities from information theory and cite some results that are needed in this paper. For a proper introduction to information theory we refer the reader to the book by Cover and Thomas [6]. In the following let h_2 denote the binary entropy function $h_2(p) = -p\log p - (1-p)\log(1-p)$ for $p \in [0,1]$. Let X, Y, and Z be random variables and let E be an event, for example the event $Y = y$. Then $\mathrm{H}(X)$ denotes the entropy of the random variable X and $\mathrm{H}(X|E)$ denotes the entropy of X with respect to the conditional distribution of X given that the event E occurred. If there are several events separated by commas, then we analogously use the conditional distribution of X given that all of the events occurred. Let $\mathrm{H}(X|Y)$ denote the conditional entropy of X given Y. Recall that $\mathrm{H}(X|Y) = \sum_y \Pr\{Y=y\}\,\mathrm{H}(X|Y=y)$. If we condition on several variables, we separate the variables by commas. If we mix events and variables in the condition, we first list the variables, after that we list the events, for example $\mathrm{H}(X|Y, Z=z)$. The mutual information of X and Y is $\mathrm{I}(X:Y) = \mathrm{H}(X) - \mathrm{H}(X|Y)$ and $\mathrm{I}(X:Y|E) = \mathrm{H}(X|E) - \mathrm{H}(X|Y,E)$ is the mutual information of X and Y with respect to the conditional distribution of X and Y given that the event E occurred. The conditional mutual information of X and Y given Z is $\mathrm{I}(X:Y|Z) = \mathrm{H}(X|Z) - \mathrm{H}(X|Y,Z)$. Recall that $\mathrm{I}(X:Y|Z) = \sum_z \Pr\{Z=z\}\,\mathrm{I}(X:Y|Z=z)$.

Suppose that the random variables X and Y have the same range. Then the Kullback Leibler distance of their distributions is $\mathrm{D}(X,Y) = \sum_v \Pr\{X=v\} \log \frac{\Pr\{X=v\}}{\Pr\{Y=v\}}$. If $\Pr\{X=v\} = 0$ in the above sum, then the corresponding term is 0 independently of the value of $\Pr\{Y=v\}$, by continuity arguments. If $\Pr\{X=v\} \neq 0$ and $\Pr\{Y=v\} = 0$ for some v, then the whole sum is defined to be equal to ∞. If E is an event, then $(X|E)$ denotes the conditional distribution of X given that the event E occurred, for example $\mathrm{D}((X|E), X)$ is the Kullback Leibler distance of the conditional distribution of X given that the event E occurred and the distribution of X. Recall that the mutual information of X and Y is the Kullback Leibler distance of the joint distribution (X,Y) and the product distribution of the marginal distributions:

$$\mathrm{I}(X:Y) = \sum_{x,y} \Pr\{X=x, Y=y\} \cdot \log \frac{\Pr\{X=x, Y=y\}}{\Pr\{X=x\} \cdot \Pr\{Y=y\}} \ .$$

The following lemma is a useful tool for the proof of lower bounds on the Kullback Leibler distance of distributions. A proof of the log sum inequality can be found in [6].

Lemma 2.1 (Log sum inequality)**.** *For nonnegative numbers a_i and b_i, where $i = 1, \ldots, n$,*

$$\sum_i a_i \log \frac{a_i}{b_i} \geq \left(\sum_i a_i \right) \log \frac{\sum_i a_i}{\sum_i b_i} \ .$$

Suppose that the random variables X and Y have the same finite range R. Then the total variation distance of their distributions is $\mathrm{V}(X,Y) = \frac{1}{2} \sum_v |\Pr\{X=v\} - \Pr\{Y=v\}|$. It is a well-known fact (see e.g. [7]) that $\mathrm{V}(X,Y) = \max_{S \subseteq R} |\Pr\{X \in S\} - \Pr\{Y \in S\}|$.

The following lemma by Kullback relates the Kullback Leibler distance of distributions to their total variation distance.

Lemma 2.2 (Kullback [8])**.** *Suppose that X and Y are random variables that have the same finite range. Then* $\mathrm{D}(X,Y) \geq 2 \cdot \mathrm{V}(X,Y)^2$.

2.3. Information Complexity

The notion of the information cost of a protocol was introduced by Chakrabarti, Shi, Wirth, and Yao [5]. The information cost of a randomized protocol is the mutual information of the input and the transcript of the protocol. Then the information complexity of a function can be defined in the canonical way. Here we will use the conditional information complexity of a function, a refinement that was introduced by Bar-Yossef, Jayram, Kumar, and Sivakumar [3].

Definition 2.3. Let B be a set, let $f\colon B^k \longrightarrow \{0,1\}$ be a function, and let $\mathbf{X} \in B^k$ and D be random variables. Suppose that P is a randomized k-party protocol for f and that $M(\mathbf{X})$ is the transcript of P for the input $\mathbf{X}$. Then the conditional information cost of P with respect to $\mathbf{X}$ and D is defined by

$$\mathrm{icost}(P;\mathbf{X}|D) = \mathrm{I}(M(\mathbf{X}) : \mathbf{X}|D) \ .$$

The conditional ε-error information complexity $\mathrm{IC}_\varepsilon(f;\mathbf{X}|D)$ of f w.r.t. $\mathbf{X}$ and D is the minimal conditional information cost of a communication protocol for $f(\mathbf{X})$ where the minimum is taken over all randomized ε-error protocols for f.

The information complexity of a function is a lower bound for the communication complexity. A proof of the next theorem can be found in [3].

Theorem 2.4. *Let B be a set, let $f\colon B^k \longrightarrow \{0,1\}$ be a function, and let $\mathbf{X} \in B^k$ and D be random variables. Then the ε-error communication complexity of f is bounded from below by* $\mathrm{IC}_\varepsilon(f;\mathbf{X}|D)$.

2.4. The Direct Sum Paradigm

Information complexity has very nice properties with respect to direct sum problems. In this section we summarize the approach of Bar-Yossef, Jayram, Kumar and Sivakumar [3] using a slightly different terminology. We call a problem f a direct sum problem if it can be decomposed into simpler problems of smaller size.

Definition 2.5. Let $f\colon (B^n)^k \longrightarrow \{0,1\}$ be a function and let $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n}) \in B^n$ for $i = 1, \ldots, k$. If there are functions $g\colon \{0,1\}^n \longrightarrow \{0,1\}$ and $h\colon B^k \longrightarrow \{0,1\}$ such that

$$f(\mathbf{x}_1, \ldots, \mathbf{x}_k) = g\left(h(x_{1,1}, x_{2,1}, \ldots, x_{k,1}) \, , \, \ldots \, , \, h(x_{1,n}, x_{2,n}, \ldots, x_{k,n}) \right)$$

then the function f is called a g-h-direct sum.

Here the goal is to express a lower bound on the conditional information complexity of f in terms of the conditional information complexity of the simpler function h and the parameter n. In order for this approach to work, the joint distribution of the inputs of h and the condition must have certain properties. As a first requirement, the condition must partition the distribution of the inputs into product distributions.

Definition 2.6. Let B be a set and let $\mathbf{X} = (X_1, \ldots, X_k) \in B^k$ and D be random variables. The variable D partitions $\mathbf{X}$, if for every d in the support of D the conditional distribution $(\mathbf{X}|D{=}d)$ is the product distribution of the distributions $(X_i|D{=}d)$ for $i = 1, \ldots, k$.

The function f can be decomposed into instances of the function h if the distribution of the inputs of f satisfies our second requirement.

Definition 2.7. Let B be a set, let $g\colon \{0,1\}^n \longrightarrow \{0,1\}$ and $h\colon B^k \longrightarrow \{0,1\}$ be functions, and let $\mathbf{X} \in B^k$ be a random variable. If for every $i \in \{1, \ldots, n\}$, for every $a \in B^k$, and for every $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in (B^k)^n$ such that $\mathbf{x}_j \in \text{support}(\mathbf{X})$ for all j

$$g\left(h(x_1), \ldots, h(x_{i-1}), h(a), h(x_{i+1}), \ldots, h(x_n)\right) = h(a)$$

then the distribution of $\mathbf{X}$ is called collapsing for g and h.

If these two requirements are met, then the conditional information complexity of f can be expressed in terms of the conditional information complexity of h and the parameter n.

Theorem 2.8 (Bar-Yossef *et al.* [3])**.** *Suppose that $f\colon (B^n)^k \longrightarrow \{0,1\}$ is a g-h-direct sum and that $\mathbf{X} \in B^k$ and D are random variables such that the distribution of $\mathbf{X}$ is collapsing for g and h and D partitions $\mathbf{X}$. Let $\mathbf{Y} = (\mathbf{Y}_1, \ldots, \mathbf{Y}_k) \in (B^n)^k$ and $\mathbf{E} \in \text{support}(D)^n$ be random variables and let Y_i^j and E^j denote the projection of $\mathbf{Y}_i$ and $\mathbf{E}$ to the jth coordinate, respectively. If the random variables $\mathbf{V}_j = ((Y_1^j, \ldots, Y_k^j), E^j)$ for $j = 1, \ldots, n$ are independent and $\mathbf{V}_j \sim (\mathbf{X}, D)$ for all j, then $\text{IC}_\varepsilon(f; \mathbf{Y}|\mathbf{E}) \geq n \cdot \text{IC}_\varepsilon(h; \mathbf{X}|D)$.*

This direct sum approach can be applied to the k-party set disjointness problem.

Observation 2.9. Let AND_ℓ and OR_ℓ denote the Boolean conjunction and disjunction of ℓ bits, respectively. Then the k-party set disjointness problem is a OR_n-AND_k-direct sum.

Consequently, for the proof of Theorem 1.2 it is sufficient to prove a lower bound on the conditional information complexity of AND_k for a distribution that satisfies the requirements of Theorem 2.8 and, in addition, honors the unique intersection promise. A distribution with these properties is defined in the following section. This approach was already used in [3] and [4].

3. The Information Complexity of AND_k

For the following distribution of D and the input $\mathbf{Z} = (Z_1, \ldots, Z_k)$ of AND_k the variable D partitions $\mathbf{Z}$ and the distribution of $\mathbf{Z}$ is collapsing for OR_n and AND_k. Additionally, there is at most a single i such that $Z_i = 1$.

Definition 3.1. From here on let $\mathbf{Z} = (Z_1, \ldots, Z_k) \in \{0,1\}^k$ and $D \in \{1, \ldots, k\}$ be random variables such that the joint distribution of $\mathbf{Z}$ and D has the following properties: D is uniformly distributed in $\{1, \ldots, k\}$. For all $i \in \{1, \ldots, k\}$ we have $\Pr\{Z_j{=}0|D{=}i\} = 1$ for $j \neq i$ and $\Pr\{Z_i{=}0|D{=}i\} = \Pr\{Z_i{=}1|D{=}i\} = \frac{1}{2}$.

Now we can state the main result of this paper, an asymptotically optimal lower bound on the information complexity of the AND_k-function for inputs that are distributed according to the last definition.

Theorem 3.2. *Let $\varepsilon < \frac{3}{10}\left(1 - \sqrt{\frac{1}{2}\log\frac{4}{3}}\right)$ be a constant. Then there is a constant $c(\varepsilon) > 0$ that does only depend on ε such that $\text{IC}_\varepsilon(\text{AND}_k; \mathbf{Z}|D) \geq c(\varepsilon)/k$.*

It is easy to see that $\mathrm{icost}(P;\mathbf{Z}|D) = 1/k$ for a trivial deterministic protocol P for AND_k where each player in turn writes his input to the blackboard until the first 0 is written. Therefore our lower bound is optimal. As we have seen, this result immediately implies Theorem 1.2, the other main result of this paper. In the rest of the paper we will outline the proof of Theorem 3.2.

3.1. Some Basic Observations

We start with some basic observations about the joint distribution of the inputs and the transcript of a protocol for AND_k with independent, uniformly distributed inputs.

Definition 3.3. From now on, let P be a fixed randomized k-player protocol that computes AND_k with error at most ε and for $\mathbf{x} \in \{0,1\}^k$ let $M(\mathbf{x})$ denote the transcript of P for the input $\mathbf{x}$. Let $\mathbf{X} = (X_1, \ldots, X_k)$ be a random variable that is uniformly distributed in $\{0,1\}^k$ and let $T = M(\mathbf{X})$ denote the transcript of P for the the input $\mathbf{X}$.

Note that the transcript $M(\mathbf{x})$ does depend on $\mathbf{x}$ and the random inputs of the players. Thus even for a fixed input $\mathbf{x}$ the transcript is a random variable whose value depends on the random bits used in the protocol.

A randomized k-party protocol can be seen as a deterministic protocol in which the ith player has two inputs: The input to the randomized protocol, in our case X_i, and as a second input the random bits that are used by the ith player. Then the first observation is a restatement of the fact that the set of the inputs (real inputs and random bits) that correspond to a fixed transcript is a combinatorial rectangle (see [9] for a definition of combinatorial rectangles).

Observation 3.4 ([3, 4])**.** Let $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^k$ and let t be an element from the support of T. Then $\Pr\{\mathbf{X}=\mathbf{x}|T=t\} = \prod_i \Pr\{X_i=x_i|T=t\}$.

We omit the simple combinatorial proof of this observation because this basic property of k-party protocols was already used in [3] and [4]. The following observation is an immediate, but very useful consequence of the previous one.

Observation 3.5. Let $\mathbf{x} = (x_1, \ldots, x_k) \in \{0,1\}^k$ and let t be an element from the support of T. Then $\Pr\{X_i=x_i|T=t, \mathbf{X}_{-i}=\mathbf{x}_{-i}\} = \Pr\{X_i=x_i|T=t\}$ for all $i \in \{1, \ldots, k\}$.

Proof. This observation follows immediately from Observation 3.4: By adding the equality from Observation 3.4 for $(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_k)$ and $(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_k)$ we obtain

$$\Pr\{\mathbf{X}_{-i}=\mathbf{x}_{-i}|T=t\} = \prod_{j\neq i} \Pr\{X_j=x_j|T=t\} \; .$$

Using this and Observation 3.4 verbatim yields

$$\begin{aligned}\Pr\{X_i=x_i|T=t, \mathbf{X}_{-i}=\mathbf{x}_{-i}\} &= \frac{\Pr\{X_i=x_i, \mathbf{X}_{-i}=\mathbf{x}_{-i}|T=t\}}{\Pr\{\mathbf{X}_{-i}=\mathbf{x}_{-i}|T=t\}} \\ &= \frac{\prod_j \Pr\{X_j=x_j|T=t\}}{\prod_{j\neq i} \Pr\{X_j=x_j|T=t\}} = \Pr\{X_i=x_i|T=t\} \; .\end{aligned}$$

■

The next observation relates the joint distribution of Z_i and $M(\mathbf{Z})$ given that $D = i$ to the joint distribution of X_i and $T = M(\mathbf{X})$ given that $\mathbf{X}_{-i} = \mathbf{0}$. Combined with the previous observations, this will be the basis for the proof of the main result.

Observation 3.6. Let $i \in \{1, \ldots, k\}$. Then $\mathrm{I}(M(\mathbf{Z}) : Z_i | D = i) = \mathrm{I}(T : X_i | \mathbf{X}_{-i} = \mathbf{0})$.

Proof. First observe that $\Pr\{\mathbf{Z} = \mathbf{v}, M(\mathbf{Z}) = t | D = i\} = \Pr\{\mathbf{X} = \mathbf{v}, T = t | \mathbf{X}_{-i} = \mathbf{0}\}$ for every $v \in \{0,1\}^k$ and every t in the support of $M(X)$ and $M(Z)$. This follows from the fact that the conditional distribution of $\mathbf{X}$ given that $\mathbf{X}_{-i} = \mathbf{0}$ is the same as the conditional distribution of $\mathbf{Z}$ given that $D = i$, the fact that the random inputs of P are independent of $\mathbf{X}$ and $\mathbf{Z}$, and the fact that the transcript is a function of the inputs and the random inputs. Then the claim of the lemma is an immediate consequence of the initial observation. ∎

3.2. Main Idea of the Proof

Like the approach of Bar-Yossef *et al.* [3], our approach is based on the observation that the distribution of the transcripts of a randomized protocol for AND_k with small error must at least be very different for the inputs $\mathbf{X} = \mathbf{0}$ and $\mathbf{X} = \mathbf{1}$. The difference is expressed using some appropriate metric on probability spaces. Then, by using Observations 3.4 and 3.5, this result is decomposed into results about the distributions of $(X_i, M(\mathbf{X}) | \mathbf{X}_{-i} = \mathbf{0})$ which are finally used to bound the conditional mutual information of $\mathbf{Z}$ and $M(\mathbf{Z})$ given D by using Observation 3.6. The result from [3] mainly uses the Hellinger distance (see [7]) to carry out this very rough outline of the proof. We will stick to the rough outline, but our result will use the Kullback Leibler distance instead of the Hellinger distance. Due to the limited space in the STACS-proceedings we can only present proof-sketches of the technical lemmas in this section. A version of this paper with full proofs can be found on the authors homepage [1].

We will first decompose the Kullback Leibler distance of the distributions $(T | \mathbf{X} = \mathbf{0})$ and $(T | \mathbf{X} = \mathbf{1})$ into results about the joint distributions of X_i and T for $i = 1, \ldots, k$. The result will be expressed in terms of the following function.

Definition 3.7. From now on, let $g(x) = x \log \frac{x}{1-x}$.

Note that the left hand side of the equation in the following lemma is the Kullback Leibler distance of $(T | \mathbf{X} = \mathbf{0})$ and $(T | \mathbf{X} = \mathbf{1})$ if S is the set of all possible transcripts.

Lemma 3.8. *Let S be a subset of the set of all possible transcripts. Then*

$$\sum_{t \in S} \Pr\{T = t | \mathbf{X} = \mathbf{0}\} \cdot \log \frac{\Pr\{T = t | \mathbf{X} = \mathbf{0}\}}{\Pr\{T = t | \mathbf{X} = \mathbf{1}\}} = 2 \sum_i \sum_{t \in S} \Pr\{T = t | \mathbf{X}_{-i} = \mathbf{0}\} \cdot g(\Pr\{X_i = 0 | T = t\}) \; .$$

Proof Sketch. The proof of this lemma is mainly based on the fact that

$$\frac{\Pr\{T = t | \mathbf{X} = \mathbf{0}\}}{\Pr\{T = t | \mathbf{X} = \mathbf{1}\}} = \frac{\Pr\{\mathbf{X} = \mathbf{0} | T = t\}}{\Pr\{\mathbf{X} = \mathbf{1} | T = t\}} \; .$$

Then Observation 3.4 can be applied to decompose the log-function into a sum. Finally, we use that $\Pr\{T = t | \mathbf{X} = \mathbf{0}\} = 2 \Pr\{T = t | \mathbf{X}_{-i} = \mathbf{0}\} \cdot \Pr\{X_i = 0 | T = t\}$ by Observation 3.5. ∎

[1] http://ls2-www.cs.uni-dortmund.de/~gronemeier/

Next, we will express a lower bound on $\mathrm{I}(M(\mathbf{Z}):Z|D)$ in terms of the following function f and set $B(\alpha)$.

Definition 3.9. From now on, let $f(x) = x\log 2x + \frac{1-x}{2}\log 2(1-x)$.

Definition 3.10. Let $B(\alpha)$ denotes the set of all transcripts t such that $\Pr\{X_i=0|T=t\} < \alpha$ for all $i \in \{1,\ldots,k\}$.

The role of the parameter α will become apparent later. The only property that is needed for the proof of the following lemma is that $\alpha > 1/2$.

Lemma 3.11. *Let $\alpha > \frac{1}{2}$ be a constant. Then*

$$\mathrm{I}(M(\mathbf{Z}):\mathbf{Z}|D) \geq \frac{1}{k}\sum_i \sum_{t\in B(\alpha)} \Pr\{T=t|\mathbf{X}_{-i}=\mathbf{0}\}\cdot f(\Pr\{X_i=0|T=t\}) \ .$$

Proof Sketch. This lemma can be proved by using that $f(x) = \frac{1}{2}(f_1(x)+f_2(x))$ where $f_1(x) = x\log 2x + (1-x)\log 2(1-x)$ and $f_2(x) = x\log 2x$. It is sufficient to prove that the lower bound holds for f_1 and f_2 instead of f. To this end one can show that

$$\mathrm{I}(M(\mathbf{Z}):\mathbf{Z}|D) = \frac{1}{k}\sum_i \sum_t \Pr\{T=t|\mathbf{X}_{-i}=\mathbf{0}\}\cdot f_1(\Pr\{X_i=0|T=t\}) \ .$$

Then the bound for f_1 is obvious since $f_1(x)$ is nonnegative for all $x \in [0,1]$. The bound for f_2 use the fact that $f_1(x) = f_2(x) + f_2(1-x)$, that $f_2(x) \geq 0$ for $x \in [1/2,1]$, and that

$$\sum_t \Pr\{T=t|\mathbf{X}_{-i}=\mathbf{0}\}\cdot f_2(\Pr\{X_i=1|T=t\})$$

is nonnegative. ∎

The right hand sides of the equation in Lemma 3.8 and the inequality in Lemma 3.11 look very similar. In fact, if there was a positive constant c such that $c\cdot f(x) \geq g(x)$ for all $x \in [0,1]$, then for a complete proof of Theorem 3.2 it would be sufficient to show that the Kullback Leibler distance of $(T|\mathbf{X}=\mathbf{0})$ and $(T|\mathbf{X}=\mathbf{1})$ is bounded from below by a constant $c(\varepsilon)$ if the error of the protocol P is bounded by ε. Unfortunately $f(x) \leq 1$ for $x \in [0,1]$ while $g(x)$ is not bounded from above for $x \in [0,1]$. So this naive first idea does not work. But the function $g(x)$ is bounded in every interval $[0,\beta]$ where $\beta < 1$. The following Lemma shows that we can easily bound $f(x)$ from below in terms of $g(x)$ if we restrict x to an appropriate interval $[0,\beta]$.

Lemma 3.12. *There is a constant $\beta > \frac{1}{2}$ such that $4\cdot f(x) \geq g(x)$ for all $x \in [0,\beta]$.*

This lemma can probably be proved in many ways. By inspection and numeric computations it is easy to verify that it holds for $\beta \approx 0.829$. Here it is more important to note that our choice of the function f is one of the crucial points of our proof: The function $g(x)$ is negative for $x \in [0,\frac{1}{2})$ and nonnegative and increasing for $x \in [\frac{1}{2},1]$. Furthermore $g(\frac{1}{2}) = 0$ and in the interval $[\frac{1}{2},1]$ the slope of $g(x)$ is bounded from below by a positive constant. It will become clear in Lemma 3.14 that we have to lower bound $f(x)$ in terms of $g(x)$ for $x \approx \frac{1}{2} + O(\frac{1}{k})$ where k is the number of players. Recall that $f(x) = \frac{1}{2}(f_1(x)+f_2(x))$ where $f_1(x) = x\log 2x + (1-x)\log 2(1-x)$ and $f_2(x) = x\log 2x$ and that we prove Lemma 3.11 by lower bounding the mutual information of $M(\mathbf{Z})$ and $\mathbf{Z}$ in terms of $f_1(x)$ and $f_2(x)$. Thus $f_1(x)$ and $f_2(x)$ would be natural candidates for the function $f(x)$. Unfortunately, neither $f_1(x)$ nor $f_2(x)$ alone does work in our proof. The function $f_1(x)$ is nonnegative for

$x \in [0,1]$, therefore $f_1(x) \geq g(x)$ for $x \in [0, \frac{1}{2}]$, but the slope of $f_1(x)$ is too small in the interval $[\frac{1}{2}, 1]$. It turns out that $f_1(\frac{1}{2} + \frac{1}{k}) \approx 1/k^2$. If we used the function $f_1(x)$ instead of $f(x)$ in our proof, we could only obtain an $\Omega(1/k^2)$ lower bound for the information complexity of AND_k. The function $f_2(x)$ does not suffer from this problem since the slope of $f_2(x)$ in $[\frac{1}{2}, 1]$ is bounded from below by a constant. But here we have the problem that $f_2(x)$ is too small for $x \in [0, \frac{1}{2})$. For every constant $c > 0$ such that $c \cdot f(x) \geq g(x)$ in the interval $x \in [\frac{1}{2}, 1]$ we have $g(x) > c \cdot f(x)$ in the interval $x \in [0, \frac{1}{2})$. Luckily, for the average $f(x)$ of $f_1(x)$ and $f_2(x)$ the good properties of the functions are preserved while the bad properties "cancel out". The bounded slope for $x \in [\frac{1}{2}, 1]$ of $f(x)$ is inherited from $f_2(x)$. The fact that $f(x)$ is not to small for $x \in [0, \frac{1}{2})$ is inherited from $f_1(x)$.

We can use the set $B(\alpha)$ in Lemma 3.11 and the set S in Lemma 3.8 to restrict t to the transcripts that satisfy $\Pr\{X_i = 0 | T = t\} \leq \beta$ for all $i \in \{1, \ldots, k\}$. Then, by our previous observations, it is easy to lower bound $f(\Pr\{X_i = 0 | T = t\})$ in terms of $g(\Pr\{X_i = 0 | T = t\})$.

Definition 3.13. Let β be the constant from Lemma 3.12. recall that $B(\alpha)$ denotes the the set of all transcripts t such that $\Pr\{X_i = 0 | T = t\} < \alpha$ for all $i \in \{1, \ldots, k\}$. Then B is a shorthand notation for the set $B(\beta)$.

Unfortunately, the restriction of t to the set $S = B$ complicates the proof of a lower bound for the left hand sum in Lemma 3.8 since we remove the largest terms from the sum. For example, we will see in the proof of Corollary 3.17 that for zero-error protocols the set B does only contain transcripts for the output 1. Therefore, by the zero-error property, $\Pr\{T \in B | \mathbf{X} = \mathbf{0}\} = 0$ for zero error protocols and the left hand sum in Lemma 3.8 is equal to 0. Consequently, without further assumptions that do not hold in general it is impossible to prove large lower bounds on the sum in Lemma 3.8 for the set $S = B$. However, the next Lemma shows that we can lower bound the sum, if we assume that $\Pr\{T \in B | \mathbf{X} = \mathbf{0}\}$ is sufficiently large.

Lemma 3.14. *Suppose that* $\Pr\{T \in B | \mathbf{X} = \mathbf{0}\} \geq \frac{3}{4}$ *and that the error* ε *of the protocol* P *is bounded by* $\varepsilon < \frac{3}{10}\left(1 - \sqrt{\frac{1}{2}\log\frac{4}{3}}\right)$. *Then*

$$\sum_{t \in B} \frac{\Pr\{T = t | \mathbf{X} = \mathbf{0}\}}{\Pr\{T \in B | \mathbf{X} = \mathbf{0}\}} \cdot \log \frac{\Pr\{T = t | \mathbf{X} = \mathbf{0}\}}{\Pr\{T = t | \mathbf{X} = \mathbf{1}\}} \geq \min\left\{\log\frac{3}{2}, 2\left(1 - \frac{10}{3}\varepsilon\right)^2 - \log\frac{4}{3}\right\} > 0\,.$$

Proof Sketch. For the proof of this lemma we consider two cases: If $\Pr\{T \in B | \mathbf{X} = \mathbf{1}\} < \frac{1}{2}$ then we can use the log sum inequality (Lemma 2.1) to lower bound the sum on the left hand side. If $\Pr\{T \in B | \mathbf{X} = \mathbf{1}\} \geq \frac{1}{2}$ then the error of the protocol P under the condition that $T \in B$ must be small both for the input $\mathbf{X} = \mathbf{0}$ and the input $\mathbf{X} = \mathbf{1}$. With this assumption we can lower bound the left hand side using Lemma 2.2 since in this case the total variation distance of $(T | \mathbf{X} = \mathbf{0}, T \in B)$ and $(T | \mathbf{X} = \mathbf{1}, T \in B)$ is large. ∎

Note that, by Lemma 3.8 and the fact that the slope of $g(x)$ is bounded from below by a positive constant for $x \in [1/2, 1]$, this lower bound can be met if $\Pr\{X_i = 0 | T = t\} = \frac{1}{2} + \Theta(\frac{1}{k})$ for all $i \in \{1, \ldots, k\}$ and every $t \in B$.

By Lemma 3.14, under the condition that $\Pr\{T \in B | \mathbf{X} = \mathbf{0}\} \geq \frac{3}{4}$ our initial naive plan of bounding f in terms of g does work. The details of this idea are elaborated on in the proof of Theorem 3.16. Next, we look at the case that $\Pr\{T \in B | \mathbf{X} = \mathbf{0}\}$ is small. It turns out that this assumption alone already leads to a large lower bound on $\mathrm{I}(M(\mathbf{Z}) : \mathbf{Z} | D)$.

Lemma 3.15. *Let α be a constant subject to $1/2 < \alpha \leq 1$. Then*

$$\mathrm{I}(M(\mathbf{Z}):\mathbf{Z}|D) \geq \frac{1}{2k} \cdot \Pr\{T \notin B(\alpha)|\mathbf{X}=\mathbf{0}\} \cdot (1 - h_2(\alpha)).$$

Proof Sketch. The proof of this lemma is based on the fact that, by the definition of $B(\alpha)$, under the condition that $T = t \notin B(\alpha)$ the entropy of X_i is bounded by $h_2(\alpha) < 1$ for at least one i. ∎

Now all prerequisites for a full proof of Theorem 3.2 are in place. It is implied by the following theorem because P was assumed to be an arbitrary ε-error protocol for AND_k.

Theorem 3.16. *Let $\varepsilon < \frac{3}{10}\left(1 - \sqrt{\frac{1}{2}\log\frac{4}{3}}\right)$ be a constant. If the error of the protocol P is bounded by ε, then there is a constant $c(\varepsilon) > 0$ that does only depend on ε such that*

$$\mathrm{I}(M(\mathbf{Z}):\mathbf{Z}|D) \geq \frac{c(\varepsilon)}{k}.$$

Proof. Recall that B is the set of all transcripts t such that $\Pr\{X_i=0|T=t\} < \beta$ for all $i \in \{1,\ldots,k\}$, where β is the constant from Lemma 3.12. For the proof of the lemma we will consider two cases.

For the first case, assume that $\Pr\{T \in B|\mathbf{X}=\mathbf{0}\} \leq \frac{3}{4}$. In this case we can apply Lemma 3.15 with $\alpha = \beta$ and we get

$$\mathrm{I}(M(\mathbf{Z}):\mathbf{Z}|D) \geq \frac{1}{2k}\Pr\{T \notin B|\mathbf{X}=\mathbf{0}\}(1 - h_2(\beta)) \geq \frac{1}{8k}(1 - h_2(\beta)) \,.$$

Note that in this case the lower bound does not depend on ε and that, since $\beta > 1/2$, there is a constant $c_1 > 0$ such that the right hand side of the last inequality is bounded from below by c_1/k.

For the second case, assume that $\Pr\{T \in B|\mathbf{X}=\mathbf{0}\} > \frac{3}{4}$. In this case we first apply Lemma 3.11 for $\alpha = \beta$, thus $B(\alpha) = B$, then Lemma 3.12, and finally Lemma 3.8 for the subset $S = B$ to get

$$\begin{aligned}
\mathrm{I}(M(\mathbf{Z}):\mathbf{Z}|D) &\geq \frac{1}{k}\sum_i \sum_{t\in B} \Pr\{T=t|\mathbf{X}_{-i}=\mathbf{0}\} \cdot f(\Pr\{X_i=0|T=t\}) \\
&\geq \frac{1}{4k}\sum_i \sum_{t\in B} \Pr\{T=t|\mathbf{X}_{-i}=\mathbf{0}\} \cdot g(\Pr\{X_i=0|T=t\}) \\
&= \frac{1}{8k}\sum_{t\in B} \Pr\{T=t|\mathbf{X}=\mathbf{0}\} \cdot \log\frac{\Pr\{T=t|\mathbf{X}=\mathbf{0}\}}{\Pr\{T=t|\mathbf{X}=\mathbf{1}\}} \,.
\end{aligned}$$

Then, by the assumption $\Pr\{T \in B|\mathbf{X}=\mathbf{0}\} > \frac{3}{4}$, we can apply Lemma 3.14 to obtain

$$\begin{aligned}
\mathrm{I}(M(\mathbf{Z}):\mathbf{Z}|D) &\geq \frac{1}{8k} \cdot \Pr\{T \in B|\mathbf{X}=\mathbf{0}\} \cdot \min\left\{\log\frac{3}{2}, 2\left(1 - \frac{10}{3}\varepsilon\right)^2 - \log\frac{4}{3}\right\} \\
&\geq \frac{3}{32k} \cdot \min\left\{\log\frac{3}{2}, 2\left(1 - \frac{10}{3}\varepsilon\right)^2 - \log\frac{4}{3}\right\} \,.
\end{aligned}$$

For $\varepsilon < \frac{3}{10}\left(1 - \sqrt{\frac{1}{2}\log\frac{4}{3}}\right)$ the minimum in the last inequality is a positive constant that does only depend on the constant ε. Hence, there is a constant $c_2(\varepsilon) > 0$ that does only

depend on the constant ε such that the right hand side is bounded from below by $c_2(\varepsilon)/k$. The claim of the Lemma follows from the two cases if we choose $c(\varepsilon) = \min\{c_1, c_2(\varepsilon)\}$. ∎

3.3. A Simple Lower Bound for Zero-Error Protocols

For zero-error protocols a lower bound can be proved by using only Lemma 3.15.

Corollary 3.17. *For every randomized k-player zero-error protocol with input $\mathbf{Z}$ and transcript $M(\mathbf{Z})$ the conditional information cost satisfies $\mathrm{I}(M(\mathbf{Z}):\mathbf{Z}|D) \geq 1/(2k)$.*

Proof. Consider the transcript T of the protocol P for the input $\mathbf{X}$. Then the corollary follows immediately from Lemma 3.15 if we set $\alpha = 1$: Recall that the output of the protocol can be inferred from the transcript and let $P(t)$ denote the output of the protocol P for transcript t. Suppose that $P(t) = 0$. Then $\Pr\{X_i = 0|T = t\} = 1$ for at least one i since otherwise, by Observation 3.4, $\Pr\{\mathbf{X} = \mathbf{1}|T = t\} > 0$ and under the condition $T = t$ the output of P would be wrong with a nonzero probability. Clearly this is not possible for zero-error protocols, hence $\Pr\{T \notin B(1)|P(T) = 0\} = 1$. Under the condition $\mathbf{X} = \mathbf{0}$ the output of P is 0 with probability 1, again by the zero-error property, therefore the last observation implies that $\Pr\{T \notin B(1)|\mathbf{X} = \mathbf{0}\} = 1$ and obviously $1 - h_2(1) = 1$. ∎

Acknowledgments

Thanks to Martin Sauerhoff for helpful discussions and proofreading.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. Information theory methods in communication complexity. In *Proc. of 17th CCC*, pages 93–102, 2002.

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

[4] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *Proc. of 18th CCC*, pages 107–117, 2003.

[5] A. Chakrabarti, Y. Shi, A. Wirth, and A. C. Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *Proc. of 42nd FOCS*, pages 270–278, 2001.

[6] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[7] A. L. Gibbs and F. E. Su. On choosing and bounding probability metrics. *International Statistical Review*, 70:419, 2002.

[8] S. Kullback. A lower bound for discrimination information in terms of variation. *IEEE Trans. Inform. Theory*, 4:126–127, 1967.

[9] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[10] A. C. Yao. Some complexity questions related to distributive computing (preliminary report). In *Proc. of 11th STOC*, pages 209–213, 1979.

Symposium on Theoretical Aspects of Computer Science year (city), pp. numbers
www.stacs-conf.org

MORE HASTE, LESS WASTE: LOWERING THE REDUNDANCY IN FULLY INDEXABLE DICTIONARIES

ROBERTO GROSSI [1] AND ALESSIO ORLANDI [1] AND RAJEEV RAMAN [2] AND S. SRINIVASA RAO [3]

[1] Dipartimento di Informatica, Università di Pisa, Italy
E-mail address: {grossi,aorlandi}@di.unipi.it

[2] Department of Computer Science, University of Leicester, United Kingdom
E-mail address: r.raman@mcs.le.ac.uk

[3] MADALGO Center*, Aarhus University, Denmark
E-mail address: ssrao@daimi.au.dk

ABSTRACT. We consider the problem of representing, in a compressed format, a bit-vector S of m bits with n **1**s, supporting the following operations, where $b \in \{\mathbf{0}, \mathbf{1}\}$:

- $\texttt{rank}_b(S, i)$ returns the number of occurrences of bit b in the prefix $S[1..i]$;
- $\texttt{select}_b(S, i)$ returns the position of the ith occurrence of bit b in S.

Such a data structure is called *fully indexable dictionary (*FID*)* [Raman, Raman, and Rao, 2007], and is at least as powerful as predecessor data structures. Viewing S as a set $X = \{x_1, x_2, \ldots, x_n\}$ of n distinct integers drawn from a universe $[m] = \{1, \ldots, m\}$, the predecessor of integer $y \in [m]$ in X is given by $\texttt{select}_1(S, \texttt{rank}_1(S, y-1))$. FIDs have many applications in succinct and compressed data structures, as they are often involved in the construction of succinct representation for a variety of abstract data types.

Our focus is on space-efficient FIDs on the RAM model with word size $\Theta(\lg m)$ and constant time for all operations, so that the time cost is independent of the input size.

Given the bitstring S to be encoded, having length m and containing n ones, the minimal amount of information that needs to be stored is $B(n, m) = \lceil \log \binom{m}{n} \rceil$. The state of the art in building a FID for S is given in [Pătraşcu, 2008] using $B(m, n) + O(m/((\log m/t)^t)) + O(m^{3/4})$ bits, to support the operations in $O(t)$ time.

Here, we propose a parametric data structure exhibiting a time/space trade-off such that, for any real constants $0 < \delta \leq 1/2$, $0 < \varepsilon \leq 1$, and integer $s > 0$, it uses

$$B(n, m) + O\left(n^{1+\delta} + n\left(\frac{m}{n^s}\right)^{\varepsilon}\right)$$

bits and performs all the operations in time $O(s\delta^{-1} + \varepsilon^{-1})$. The improvement is twofold: our redundancy can be lowered parametrically and, fixing $s = O(1)$, we get a constant-time FID whose space is $B(n, m) + O(m^{\varepsilon}/\text{poly}(n))$ bits, for sufficiently large m. This is a significant improvement compared to the previous bounds for the general case.

*Center for Massive Data Algorithmics, a center of the Danish National Research Foundation

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 517-528
http://drops.dagstuhl.de/opus/volltexte/2009/1847

1. Introduction

Data structures for dictionaries [3, 27, 34, 37], text indexing [5, 12, 22, 24, 31, 32], and representing semi-structured data [11, 14, 15, 30, 37], often require the very space-efficient representation of a bivector S of m bits with n **1**s (and $m-n$ **0**s). Since there are $\binom{m}{n}$ possible choices of n **1**s out of the m bits in S, a simple information-theoretic argument shows that we need at least $B(n,m) = \lceil \log \binom{m}{n} \rceil$ bits of space, in the worst case, to store S in some compressed format. However, for the aforementioned applications, it is not enough just to store the compressed S, as one would like to support the following operations on S, for $b \in \{\mathbf{0}, \mathbf{1}\}$:

- $\texttt{rank}_b(S,i)$ returns the number of occurrences of bit b in the prefix $S[1..i]$;
- $\texttt{select}_b(S,i)$ returns the position of the ith occurrence of bit b in S.

Our focus will be on space-efficient data structures that support these operations efficiently, on the RAM model with word size $\Theta(\log m)$. The resulting data structure is called a *fully indexable dictionary* (FID) [37] and is quite powerful. For example, S can equally represent a set $X = \{x_1, x_2, \ldots, x_n\}$ of n distinct integers drawn from a universe $[m] = \{1, \ldots, m\}$, where $S[x_i] = \mathbf{1}$, for $1 \le i \le n$, while the remaining $m-n$ bits of S are **0**s. In this context, the classical problem of finding the *predecessor* in X of a given integer $y \in [m]$ (i.e. the greatest lower bound of y in X) can be solved with two FID queries on S by $\texttt{select}_1(S, \texttt{rank}_1(S, y-1))$. FIDs have also connections with coding theory, since they represent a sort of locally decodable source code for S [4]. They are at the heart of compressed text indexing since they enable space to be squeezed down to the high-order entropy when properly employed [20]. Finally, they are the building blocks for many complex low space data structures [2, 9, 28, 29] that require $O(1)$ lookup time, namely, their time complexity is independent of the number of entries stored at the expense of using some extra space.

To support the `rank` and `select` operations in $O(t)$ time, for some parameter t, it appears to be necessary to use additional space, beyond the bound $B(n,m)$ needed for representing the bitstring S in compressed format. This extra space is termed the *redundancy* $R(n,m,t)$ of the data structure, and gives a total of $B(n,m)+R(n,m,t)$ bits [13]. Although the leading term $B(n,m)$ is optimal from the information-theoretic point of view, a discrepancy between theory and practice emerges when implementing FIDs for various applications [6, 19, 21, 23, 33, 39]. In particular, the term $B(n,m)$ is often of the same order as, if not superseded by, the redundancy term $R(n,m,t)$. For example, consider a constant-time FID storing $n = o(m/\text{polylog}(m))$ integers from the universe $[m]$: here, $B(n,m)$ is negligible when compared to the best known bound of $R(n,m,1) = O(m/\text{polylog}(m))$ [35].

Our goal is that of reducing the redundancy $R(n,m,t)$ for the general case $n \le m$. Although most of the previous work has generally focussed on the case $t = O(1)$, and $m = n \cdot \text{polylog}(n)$, the burgeoning range of applications (and their complexity) warrant a much more thorough study of the function $R(n,m,t)$.

There are some inherent limitations on how small can the redundancy $R(n,m,t)$ be, since FIDs are connected to data structures for the predecessor problem, and we can inherit the predecessor lower bounds regarding several time/space tradeoffs. The connection between FIDs and the predecessor problem is well known [1, 23, 36, 37] and is further developed in this paper, going beyond the simple inheritance of lower bounds. A predecessor data structure which gives access to the underlying data set is, informally, a way to support *half* the operations natively: either $\texttt{select}_1$ and $\texttt{rank}_1$, or $\texttt{select}_0$ and $\texttt{rank}_0$. In fact, we show that a data structure solving the predecessor problem can be turned into a FID and

can also be made to store the data set using $B(n,m)+O(n)$ bits, under certain assumptions over the data structure.

Consequently, if we wish to understand the limitations in reducing the redundancy $R(n,m,t)$ of the space bounds for FIDs, we must briefly survey the state of the art for the lower bounds involving the predecessor problem. The work in [36] shows a number of lower bounds and matching upper bounds for the predecessor problem, using data structures occupying at least $\Omega(n)$ words, from which we obtain, for example, that $R(n,m,1)$ can be $o(n)$ only when $n = \text{polylog}(m)$ (a degenerate case) or $m = n\,\text{polylog}(n)$. For $m = n^{O(1)}$, the lower bound for $B(n,m) + R(n,m,1)$ is $\Omega(n^{1+\delta})$ for any fixed constant $\delta > 0$. Note that in the latter case, $B(n,m) = O(n \log m) = o(R(n,m,1))$, so the "redundancy" is larger than $B(n,m)$. Since $\texttt{rank}_1$ is at least as hard as the predecessor problem, as noted in [1, 36], then all FIDs suffer from the same limitations. (It is obvious that $\texttt{rank}_0$ and $\texttt{rank}_1$ have the same complexity, as $\texttt{rank}_0(S,i) + \texttt{rank}_1(S,i) = i$.) As noted in [37, Lemma 7.3], $\texttt{select}_0$ is also at least as hard as the predecessor problem. Other lower bounds on the redundancy were given for "systematic" encodings of S (see [13, 16, 26] and related papers), but they are not relevant here since our focus is on "non-systematic" encodings [17, 18], which have provably lower redundancy. (In "non-systematic" encodings one can store S in compressed format.)

In terms of upper bounds for $R(n,m,t)$, a number are known, of which we only enumerate the most relevant here. For systematic structures, an optimal upper bound is given by [16] for $R(n,m,O(1)) = O(m \log\log m/\log m)$. Otherwise, a very recent upper bound in [35] gives $R(n,m,t) = O(m/((\log m)/t)^t + m^{3/4}\text{polylog}(m))$ for any constant $t > 0$. These bounds are most interesting when $m = n \cdot \text{polylog}(n)$. As noted earlier, sets that are sparser are worthy of closer study. For such sets, one cannot have best of two worlds: one would either have to look to support queries in non-constant time but smaller space, or give up on attaining $R(n,m,1) = o(B(n,m))$ for constant-time operations.

The main role of generic case FIDs is expressed when they take part in more structured data structures (e.g. succinct trees) where there is no prior knowledge of the relationship between n and m. Our main contribution goes along this path, striving for constant-time operations. Namely, we devise a constant-time FID having redundancy $R(n,m,O(1)) = O(n^{1+\delta} + n(m/n^s)^\varepsilon)$, for any fixed constants $\delta < 1/2$, $\varepsilon < 1$ and $s > 0$ (Theorem 3.1). The running time of the operations is always $O(1)$ for $\texttt{select}_1$ (which is insensitive to time-space tradeoffs) and is $O(\varepsilon^{-1} + s\delta^{-1}) = O(1)$ for the remaining operations. When m is sufficiently large, our constant-time FID uses just $B(n,m) + O(m^\varepsilon/\text{poly}(n))$ bits, which is a significant improvement compared to the previous bounds for the general case, as we move from a redundancy of kind $O(m/\text{polylog}(m))$ to a one of kind $O(m^\varepsilon)$, by proving for the first time that polynomial reduction in space is possible.

Moreover, when instantiated in a polynomial universe case (when $m = \Theta(n^{O(1)})$, for a sufficiently small ε, the redundancy is dominated by $n^{1+\delta}$, thus extending the known predecessor search data structure with all four FID operations without using a second copy of the data. Otherwise, the m^ε term is dominant when the universe is superpolynomial, e.g. when $m = \Theta(2^{\log^c n})$ for $c > 1$. In such cases we may not match the lower bounds for predecessor search; however, this is the price for a solution which is agnostic of m, n relationship.

We base our findings on the Elias-Fano encoding scheme [7, 8], which gives the basis for FIDs naturally supporting $\texttt{select}_1$ in $O(1)$ time.

2. Elias-Fano Revisited

We review how the Elias-Fano scheme [7, 8, 33, 39] works for an arbitrary set $X = \{x_1 < \cdots < x_n\}$ of n integers chosen from a universe $[m]$. Recall that X is equivalent to its characteristic function mapped to a bitstring S of length m, so that $S[x_i] = \mathbf{1}$ for $1 \le i \le n$ while the remaining $m-n$ bits of S are $\mathbf{0}$s. Based on the Elias-Fano encoding, we will describe the main ideas behind our new implementation of fully indexable dictionaries (FIDs). We also assume that $n \le m/2$—otherwise we build a FID on the complement set of X (and still provide the same functionalities), which improves space consumption although it does not guarantee $\texttt{select}_1$ in $O(1)$ time.

Elias-Fano encoding. Let us arrange the integers of X as a *sorted* sequence of consecutive words of $\log m$ bits each. Consider the first[1] $\lceil \log n \rceil$ bits of each integer x_i, called h_i, where $1 \le i \le n$. We say that any two integers x_i and x_j belong to the same *superblock* if $h_i = h_j$.

The sequence $h_1 \le h_2 \le \cdots \le h_n$ can be stored as a bitvector H in $3n$ bits, instead of using the standard $n\lceil \log n \rceil$ bits. It is the classical unary representation, in which an integer $x \ge 0$ is represented with x $\mathbf{0}$s followed by a $\mathbf{1}$. Namely, the values $h_1, h_2 - h_1, \ldots, h_n - h_{n-1}$ are stored in unary as a multiset. For example, the sequence $h_1, h_2, h_3, h_4, h_5 = 1, 1, 2, 3, 3$ is stored as $H = \mathbf{01101011}$, where the ith $\mathbf{1}$ in H corresponds to h_i, and the number of $\mathbf{0}$s from the beginning of H up to the ith $\mathbf{1}$ gives h_i itself. The remaining portion of the original sequence, that is, the last $\log m - \lceil \log n \rceil$ bits in x_i that are not in h_i, are stored as the ith entry of a simple array L. Hence, we can reconstruct x_i as the concatenation of h_i and $L[i]$, for $1 \le i \le n$. The total space used by H is at most $2^{\lceil \log n \rceil} + n \le 3n$ bits and that used by L is $n \times (\log m - \lceil \log n \rceil) \le n \log(m/n)$ bits.

Interestingly, the plain storage of the bits in L is related to the information-theoretic minimum, namely, $n \log(m/n) \le B(n, m)$ bits, since for $n \le m/2$, $B(n, m) \sim n \log(m/n) + 1.44\, n$ by means of Stirling approximation. In other words, the simple way of representing the integers in X using Elias-Fano encoding requires at most $n \log(m/n) + 3n$ bits, which is nearly $1.56\, n$ away from the theoretical lower bound $B(n, m)$. If we employ a constant-time FID to store H, Elias-Fano encoding uses a total of $B(n, m) + 1.56\, n + o(n)$ bits.

Rank and select operations vs predecessor search. Using the available machinery—the FID on H and the plain array L—we can perform $\texttt{select}_1(i)$ on X in $O(1)$ time: we first recover $h_i = \texttt{select}_1(H, i) - i$ and then concatenate it to the fixed-length $L[i]$ to obtain x_i in $O(1)$ time [22]. As for $\texttt{rank}$ and $\texttt{select}_0$, we point out that they are intimately related to the *predecessor search*, as we show below (the converse has already been pointed out in the Introduction).

Answering $\texttt{rank}_1(k)$ in X is equivalent to finding the predecessor x_i of k in X, since $\texttt{rank}_1(k) = i$ when x_i is the predecessor of k. Note that $\texttt{rank}_0(k) = k - \texttt{rank}_1(k)$, so performing this operation also amounts to finding the predecessor. As for $\texttt{select}_0(i)$ in X, let $\overline{X} = [m] \setminus X = \{v_1, v_2, \ldots, v_{m-n}\}$ be the complement of X, where $v_i < v_{i+1}$, for $1 \le i < m - n$. Given any $1 \le i \le m - n$, our goal is to find $\texttt{select}_0(i) = v_i$ in constant time, thus motivating that our assumption $n \le m/2$ is w.l.o.g.: whenever $n \le m/2$, we store the complement set of X and swap the zero- and one-related operations.

The key observation comes from the fact that we can associate each x_l with a new value $y_l = \big|\{v_j \in \overline{X} \text{ such that } v_j < x_l\}\big|$, which is the number of elements in $\overline{X}$ that precede x_l,

[1] Here we use Elias' original choice of ceiling and floors, thus our bounds slightly differ from the *sdarray* structure of [33], where they obtain $n\lceil \log(m/n) \rceil + 2n$.

where $1 \leq l \leq n$. The relation among the two quantities is simple, namely, $y_l = x_l - l$, as we know that exactly $l-1$ elements of X precede x_l and so the remaining elements that precede x_l must originate from $\overline{X}$. Since we will often refer to it, we call the set $Y = \{y_1, y_2, \ldots, y_n\}$ the *dual representation* of the set X.

Returning to the main problem of answering $\texttt{select}_0(i)$ in X, our first step is to find the predecessor y_j of i in Y, namely, the largest index j such that $y_j < i$. As a result, we infer that x_j is the predecessor of the *unknown* v_i (which will be our answer) in the set X. We now have all the ingredients to deduce the value of v_i. Specifically, the y_jth element of $\overline{X}$ occurs before x_j in the universe, and there is a nonempty run of elements of X up to and including position x_j, followed by $i - y_j$ elements of $\overline{X}$ up to and including (the unknown) v_i. Hence, $v_i = x_j + i - y_j$ and, since $y_j = x_j - j$, we return $v_i = x_j + i - x_j + j = i + j$. (An alternative way to see $v_i = i + j$ is that $x_1, x_2, \ldots, x_j$ are the only elements of X to the left of the unknown v_i.) We have thus proved the following.

Lemma 2.1. *Using the Elias-Fano encoding, the* $\texttt{select}_1$ *operation takes constant time, while the* `rank` *and* $\texttt{select}_0$ *operations can be reduced in constant time to predecessor search in the sets* X *and* Y*, respectively.*

The following theorem implies that we can use both lower and upper bounds of the predecessor problem to obtain a FID, and vice versa. Below, we call a data structure storing X *set-preserving* if it stores $x_1, \ldots, x_n$ *verbatim* in a contiguous set of memory cells.

Theorem 2.2. *For a given set* X *of* n *integers over the universe* $[m]$*, let* $\text{FID}(t, s)$ *be a* FID *that takes* t *time and* s *bits of space to support* `rank` *and* `select`*. Also, let* $\text{PRED}(t, s)$ *be a static data structure that takes* t *time and* s *bits of space to support predecessor queries on* X*, where the integers in* X *are stored in sorted order using* $n \log m \leq s$ *bits. Then,*

1. *given a* $\text{FID}(t, s)$*, we can obtain a* $\text{PRED}(O(t), s)$*;*
2. *given a set-preserving* $\text{PRED}(t, s)$*, we can obtain a* $\text{FID}(O(t), s - n \log n + O(n))$ *(equivalently,* $R(n, m, t) = s - n \log m + O(n)$*) with constant-time* $\texttt{select}_1$*.*
3. *if there exists a non set-preserving* $\text{PRED}(t, s)$*, we can obtain a* $\text{FID}(O(t), 2s + O(n))$ *with constant-time* $\texttt{select}_1$*.*

Proof (sketch). The first statement easily follows by observing that the predecessor of k in X is returned in $O(1)$ time by $\texttt{select}_1(S, \texttt{rank}_1(S, k-1))$, where S is the characteristic bitstring of X. Focusing on the second statement, it suffices to encode X using the Elias Fano encoding, achieving space $s - n \log n + O(n)$.

To further support $\texttt{select}_0$, we exploit the properties of Y and X. Namely, there exists a maximal subset $X' \subseteq X$ so that its dual representation Y' is strictly increasing, thus being searchable by a predecessor data structure. Hence we split X into X' and the remaining subsequence X'' and produce two Elias-Fano encodings which can be easily combined by means of an extra $O(n)$ bits FID in order to perform $\texttt{select}_1$, $\texttt{rank}_1$ and $\texttt{rank}_0$. $\texttt{select}_0$ can be supported by exploiting the set preserviness of the data structure, thus building only the extra data structure to search Y' and not storing Y'. When data structures are not set-preserving, we simply replicate the data and store Y', thus giving a justification to the $O()$ factor.

■

3. Basic Components and Main Result

We now address and solve two questions, which are fundamental to attain a $O(t)$-time FID with $B(n,m)+R(n,m,t)$ bits of storage using Lemma 2.1 and Theorem 2.2: (1) how to devise an efficient index data structure that can implement predecessor search using Elias-Fano representation with tunable time-space tradeoff, and (2) how to keep its redundancy $R(n,m,t)$ small.

Before answering the above questions, we give an overview of the two basic tools that are adopted in our construction (the string B-tree [10] and a modified van Emde Boas tree [36, 38]). We next develop our major ideas that, combined with these tools, achieve the desired time-space tradeoff, proving our main result.

Theorem 3.1. *Let $s > 0$ be an integer and let $0 \leq \varepsilon, \delta \leq 1$ be reals. For any bitstring S, $|S| = m$, having cardinality n, there exists a fully indexable dictionary solving all operations in time $O(s\delta^{-1} + \varepsilon^{-1})$ using $B(n,m) + O(n^{1+\delta} + n(m/n^s)^{\varepsilon})$ bits of space.*

Modified van Emde Boas trees. Pătraşcu and Thorup [36] have given some matching upper and lower bounds for the predecessor problem. The discussion hereafter regards the second branch of their bound: as a candidate bound they involve the equation (with our terminolgy and assuming our word RAM model) $t = \log(\log(m/n)/\log(z/n))$, where t is our desired time bound and z is the space in bits. By reversing the equation and setting $\epsilon = 2^{-t}$, we obtain $z = \Theta(n(m/n)^{\epsilon})$ bits. As mentioned in [36], the tradeoff is tight for a polynomial universe $m = n^{\gamma}$, for $\gamma > 1$, so the above redundancy cannot be lower than $\Theta(n^{1+\delta})$ for any fixed $\delta > 0$.

They also describe a variation of van Emde Boas (VEB) trees [38] matching the bound for polynomial universes, namely producing a data structure supporting predecessor search that takes $O(\log \frac{\log(m/n)}{\log(z/n)})$ time occupying $O(z \log m)$ bits. In other words, for constant-time queries, we should have $\log(m/n) \sim \log(z/n)$, which implies that the space is $z = \Theta(n(m/n)^{\epsilon})$. They target the use of their data structure for polynomial universes, since for different cases they build different data structures. However, the construction makes no assumption on the above relation and we can extend the result to arbitrary values of m. By Theorem 2.2, we can derive a constant-time FID with redundancy $R(n,m,O(1)) = O(n(m/n)^{\epsilon})$.

Corollary 3.2. *Using a modified VEB tree, we can implement a FID that uses $B(n,m) + O(n(m/n)^{\varepsilon})$ bits of space, and supports all operations in $O(\log(1/\varepsilon))$ time, for any constant $\varepsilon > 0$.*

The above corollary implies that we can obtain a first polynomial reduction by a straightforward application of existing results. However, we will show that we can do better for sufficiently large m, and effectively reduce the term $n(m/n)^{\varepsilon}$ to $n^{1+\delta} + n(m/n^s)^{\varepsilon}$. The rest of the paper is devoted to this goal.

String B-Tree: blind search for the integers. We introduce a variant of string B-tree to support predecessor search in a set of integers. Given a set of integers $X = \{x_1, \ldots, x_p\}$ from the universe $[u]$, we want obtain a space-efficient representation of X that supports predecessor queries efficiently. We develop the following structure:

Lemma 3.3. *Given a set X of p integers from the universe $[u]$, there exists a representation that uses extra $O(p \log\log u)$ bits apart from storing the elements of X, that supports*

predecessor queries on X in $O(\log p/\log\log u)$ time. The algorithm requires access to a precomputed table of size $O(u^\gamma)$ bits, for some positive constant $\gamma < 1$, which can be shared among all instances of the structure with the same universe size.

Proof. The structure is essentially a succinct version of string B-tree on the elements of X interpreted as binary strings of length $\log u$, with branching factor $b = O(\sqrt{\log u})$. Thus, it is enough to describe how to support predecessor queries in a set of b elements in constant time, and the query time follows, as the height of the tree is $O(\log p/\log\log u)$. Given a set $x_1, x_2, \ldots, x_b$ of integers from $[u]$ that need to be stored at a node of the string B-tree, we construct a compact trie (Patricia trie) over these integers (interpreted as binary strings of length $\log u$), having b leaves and $b-1$ internal nodes. The leaves disposition follows the sorting order of X. Each internal node is associated with a *skip value*, indicating the string depth at which the LCP with previous string ends. Canonically, left-pointing edges are labeled with a **0** and right-pointing with a **1**. Apart from storing the keys in sorted order, it is enough to store the tree structure and the skip values of the edges. This information can be represented using $O(b\log\log u)$ bits, as each skip value is at most $\log u$ and the trie is represented in $O(b)$ bits.

Given an element $y \in [u]$, the search for the predecessor of y proceeds in two stages. In the first stage, we simply follow the compact trie matching the appropriate bits of y to find a leaf v. Let x_i be the element associated with leaf v. One can show that x_i is the key that shares the longest common prefix with y among all the keys in X. In the second stage, we compare y with x_i to find the longest common prefix of y and x_i (which is either the leftmost or rightmost leaf of the internal node at which the search ends). By following the path in the compact trie governed by this longest common prefix, one can find the predecessor of y in X. We refer the reader to [10] for more details and the correctness of the search algorithm. The first stage of the search does not need to look at any of the elements associated with the leaves. Thus this step can be performed using a precomputed table of size $O(u^\gamma)$ bits, for some positive constant $\gamma < 1$ (by dividing the binary representation of y into chunks of size smaller than $\gamma\log u$ bits each). In the second stage, finding the longest common prefix of y and x_i can be done using bitwise operations. We again use the precomputed table to follow the path governed by the longest common prefix, to find the predecessor of y. ∎

4. Main Ideas for Achieving Polynomial Redundancy

In this section, we give a full explanation of the main result, Theorem 3.1. We first give an overview, and then detail the multiranking problem by illustrating remaining details involving the construction of our data structure.

4.1. Overview of our recursive dictionary

We consider the $\texttt{rank}_1$ operation only, leaving the effective development of the details to the next sections. A widely used approach to the FID problem (e.g. see [25, 27]) lies in splitting the universe $[m]$ into different chunks and operating independently in each chunk, storing the rank at the beginning of the block. Queries are redirected into a chunk via a preliminary *distributing* data structure and the *local* data structure is used to solve it. Thus, the space occupancy is the distributing structure (once) plus all chunks. Our approach is

orthogonal, and it guarantees better control of the parameter of subproblems we instantiate with respect to many previous approaches.

Let X ($|X| = n$) be the integer sequence of values drawn from $[m]$ and let $q \in [m]$ be a generic rank query. Our goal is to produce a simple function $f : [m] \to [m/n]$ and a machinery that generates a sequence $\tilde{X}$ from X of length n coming from the universe $[m/n]$, so that given the predecessor of $\tilde{q} = f(q)$ in $\tilde{X}$, we can recover the predecessor of q in X. By this way, we can reduce recursively, multiple times, the rank problem while keeping a single sequence per step, instead of having one data structure per chunk.

Easily enough, f is the "cutting" operation of the upper $\log n$ bits operated by the Elias Fano construction, which generates p different superblocks. Let $X_1^l, \ldots, X_p^l$ the sets of lower $\log(m/n)$ bits of values in X, one per superblock. We define our $\tilde{X}$ as $\tilde{X} = \cup_{1 \le i \le p} X_i^l$, that is, the set of unique values we can extract from the X^ls. Suppose we have an oracle function ψ, so that given a value $\tilde{x} \in \tilde{X}$ and an index $j \in [p]$, $\psi(j, \tilde{x})$ is the predecessor of $\tilde{x}$ in X_j^l. We also recall from Section 2 that the upper bit vector H of the Elias Fano construction over X can answer the query $\texttt{rank}_1(x/2^{\lceil \log n \rceil})$ in constant time (by performing $\texttt{select}_0(H, x/2^{\lceil \log n \rceil})$. That is, it can give the rank value at the beginning of each superblock.

Given a query q we can perform $\texttt{rank}_1(q)$ in the following way: we use H to reduce the problem within the superblock and know the rank at the beginning of the superblock j. We then have the lower bits of our query ($f(q)$) and the sequence $\tilde{X}$: we rank $f(q)$ there, obtaining a certain result, say v; we finally refer to our oracle to find the predecessor of v into X_j^l, and thus find the real answer for $\texttt{rank}_1(q)$. The main justification of this architecture is the following: in any superblock, the predecessor of some value can exhibit only certain values in its lower bits (those in $\tilde{X}$), thus once given the predecessor of $f(q)$ our necessary step is only to reduce the problem within $[|\tilde{X}|]$ as the lower bits for any superblock are a subset of $\tilde{X}$. The impact of such choice is, as explained later, to let us implement the above oracle in just $O(n^{1+\delta})$ bits, for any $0 < \delta < 1$. That is, by using a superlinear number of bits in n, we will be able to let m drop polynomially both in n and m.

The above construction, thus, requires one to write X in an Elias Fano dictionary, plus the oracle space and the space to solve the predecessor problem on $\tilde{X}$. The first part accounts for $B(n, m) + O(n)$ bits, to which we add $O(n^{1+\delta})$ bits for the oracle. By carefully employing the String B-tree we can shrink the number of elements of $\tilde{X}$ to $O(n/\log^2 n)$ elements, leaving us with the problem of ranking on a sequence of such length and universe $[m/n]$. We solve the problem by replicating the entire schema from the beginning. Up to the final stage of recursion, the series representing the space occupancy gives approximately $O((n \log(m/n))/\log^{2i} n + (n/\log^{2i} n)^{1+\delta})$ bits at the i-th step, descending geometrically. Each step can be traversed in constant time during a query, so the overall time is constant again. More interestingly, at each step we reduce the universe size of the outcoming sequence to mn^{-i}. Thus, at the final step s, we employ the previous result of Corollary 3.2 and obtain a final redundancy of $O(m^{\varepsilon} n^{1-s\varepsilon})$.

4.2. Multiranking

We now give further details on our construction. Mainly, we show that using our choice on how to build $\tilde{X}$ and the function f, being able to rank over $\tilde{X}$ we can build the oracle in $O(n^{1+\delta})$ bits. We do it by illustrating, in a broader framework, the multiranking problem.

We are given a *universe* $[u]$ (in our dictionary case, we start by setting $u = m$), and a set of nonempty sequences $A_1, \ldots, A_c$ each containing a sorted subset of $[u]$. We also define $r = \sum_{1 \leq j \leq n} |A_j|$ as the global number of elements. The goal is, given two values $1 \leq i \leq c$ (the wanted superblock $\hat{s}$) and $1 \leq q \leq u$ (the query $f(q)$), perform $\texttt{rank}_1(q)$ in the set A_i (in our case, the head in $\hat{s}$ that is predecessor of the searched key) in $O(1)$ time and small space.

A trivial solution to this problem would essentially build a FID for each of the sequences, thus spending a space proportional to $O(cu)$, which is prohibitive. Instead, we can carefully exploit the global nature of this task and solve it in less space. The core of this technique is the *universe scaling* procedure. We perform the union of all the A sequences and extract a new, single sequence Λ containing only the distinct values that appear in the union (that is, we kill duplicates). Λ is named the *alphabet* for our problem and we denote its length with $t \leq r$. Next, we rewrite all sequences by using rank of their elements in the alphabet instead of the initial arguments: now each sequence is defined on $[t]$.

The multiranking problem is solved in two phases. We first perform ranking of the query q on Λ and then we exploit the information to recover the predecessor in the given set. Here we achieve our goal to *(i)* decouple a phase that depends on the universe from one that dependes on the elements and *(ii)* have only one version of the problem standing on the initial universe. The following lemma solves the multiranking problem completely, that is, outside our original distinction between a oracle and the alphabet ranking:

Lemma 4.1. *There exists a data structure solving the* `multirank` *problem over c nonempty increasing sequences $\mathbb{A} = \{A_1, \ldots, A_c\}$ with elements drawn from the universe $[u]$, having r elements in total using $B(r, u) + O(r^{1+\delta}) + o(u)$ bits for any given $0 < \delta < 1/2$.*

Proof. Let Λ be the alphabet defined over u by the sequences in $\mathbb{A}$, and let $t = |\Lambda|$. For each of the sequences in $\mathbb{A}$ we create a bitvector β_i of length t where the $\beta_{ij} = \mathbf{1}$ if $\Lambda_j \in A_i$. We first view β_is as rows of a matrix of size tc; since $t \leq r$ and each of the sequences are non-empty (and hence $r \geq c$), the matrix is of size $O(r^2)$. We linearize the matrix by concatenating its rows and obtain a new bitvector β' on which we want to perform predecessor search. We note that the universe size of this bitvector is $O(r^2)$, that is, the universe is polynomial. We store β' using the data structure of Corollary 3.2 setting the time to $\log(1/\delta)$, so that space turns out to be $O(r^{1+\delta})$. Finally, we store we store a FID occupying $B(r, u) + o(u)$ that represents the subset Λ of the universe $[u]$.

Solving the multirank is easy now: given a query q and a set index i, we use the $o(u)$ FID and find $\lambda = \texttt{rank}_1(q)$ in U, which leads to the predecessor into the alphabet Λ of our query q. Since $\lambda \in [t]$ we can now use the β FID to find $p = \texttt{rank}_1(ti + \lambda)$. The final answer is clearly $p - \texttt{rank}_1(ti)$. ∎

4.3. Completing the puzzle

The multiranking problem is closely connected with the Elias-Fano representation of Section 2. When plugged in our framework, as explained in Section 4.1, that we can use our data structure itself to implement the ranking procedure. Similarly we can use it for $\texttt{select}_0$ by employing another set of data.

We are left with just one major detail. Each time we produce the output sequence $\tilde{X}$, containing the lower bits for all elements, our only clue for the number of elements is the worst case upper bound n, which is unacceptable. We now review the whole construction

and employ the string B-tree to have a polylogarithmic reduction on the number of elements, paying $O(n \log\log m)$ bits per recursion step. Generally, at each step we receive a sequence X_i as input and must output a new sequence X_{i+1} plus some data structures that can link the predecessor problem for X_i to X_{i+1}. Each X_i is stored in an Elias-Fano dictionary, and the sets of superblocks and lower bits sequences are built as explained before. We then apply a further reduction step on the problem cardinality. Each superblock can be either *slim* or *fat* depending on whether it contains less than $\log^2 n$ elements or not. Each superblock is split into blocks of size $\log^2 n$, apart from the last block, and for each block we store a String B-tree with fan-out $\sqrt{\log n}$. Since the block is polylogarithmic in size, by means of shared precomputed tables we can perform predecessor search in constant time. Slim superblocks are handled directly by the tree and they do not participate further in the construction. For each block in a fat superblock, we logically extract its *head*, that is, the smallest element in it. We now use heads in the multiranking problems and we build the output sequence X_{i+1} using only heads lower bits. As there can only be at most $O(n/\log^2 n)$ blocks in fat superblocks, the size of the output sequence is at most $O(n/\log^2 n)$. The oracle is built as usual, on the heads, using $O(n^{1+\delta})$ bits.

Ranking now performs the following steps: for each recursive step, it uses the Elias-Fano H vector to move into a superblock and at the same time check if it is slim or fat. In the latter case, it first outsources the query for the lower bits to the next dictionary, then feeds the answer to the multiranking instance and returns the actual answer. Thus, we just proved the following (with $v = \log^2 n$ and $w = n$):

Theorem 4.2. *Let w and v be two integer parameters and let $0 < \delta < 1/2$ be a real constant. Given $X_i, n_i \geq v$ and $m_i > w$, where $n_i \leq m_i$, there exists a procedure that produces a data structure involved in predecessor search. The data structure occupies $B(n_i, m_i) + O(w + n_i \log\log m_i + n_i^{1+\delta})$ space, and in $O(\delta^{-1})$ time, it reduces a predecessor query on X_i to a predecessor query on a new sequence X_{i+1} of length $n_{i+1} = O(n_i/v)$ over a universe $[m_{i+1}]$, where $m_{i+1} = m_i/w$.*

We must then deal with the last two steps. The first step aims at supporting $\texttt{select}_0$ since the above data structure can only support $\texttt{rank}_1$. The second step deals with how treat the final sequence after a number of iteration steps have been executed. We can finally give the proof of our main result:

Proof of Theorem 3.1. Let $X \subseteq [m]$ be the set whose characteristic vector is S. The data structure involves recursive instances of Theorem 4.2, by starting with $X_0 = X$ and using each step's output as input for the next step. As previously mentioned, we must only cover the base case and the last recursive step. We begin by describing the whole data structure, moving to algorithms later on. We start by partitioning X into X' and X'' as described in the proof of Theorem 2.2, so that the construction is operated on both X' and X''. We now describe representation of X'; X'' is stored in a similar way. We recursively build smaller sequences by invoking Theorem 4.2 exactly s times, using δ as given, and parameters $w = n$, $v = \log^2 m$. By invoking Corollary 3.2 the space bound easily follows. To support $\texttt{select}_0$ on the original sequence, we operate on the X' sequence alone, since when transformed to its dual Y', we obtain a strictly monotone sequence. Interpreting X' as an implicit representation of Y' we build a multiset representation for the high bits (H'), a new set of succinct string B-trees using the superblocks of the dual sequence and thought of as operating on Y' (similarly to Theorem 2.2) and a new set of s recursive applications of Theorem 4.2.

$\texttt{select}_1$ is trivial, thanks to the machinery of Theorem 2.2. The $\texttt{rank}_1$ algorithm for a query q is performed on both X' and X'' FID: we start by querying H_0, the upper bits of F'_0 (F''_0 respectively) for $q/2^{\lceil \log n \rceil}$, thus identifying a certain superblock in which the predecessor for q can appear. Unless the superblock is slim (refer to proof of Theorem 4.2) we must continue to search through the next lower-order bits. This is done via multiranking, which recurses in a cascading manner with the same technique on the s steps up to the last FID, that returns the answer. The chain is then walked backwards to find the root FID representative. We finally proceed through the succinct string B-tree to find the head and the next succinct string B-tree until we find the predecessor of q. The last step for recursion takes $O(\varepsilon^{-1})$ time. All the middle steps for multiranking and succinct string B-tree traversals take $O(s\delta^{-1} + s)$ time. To support $\texttt{select}_0$, we act on X', using exactly the same algorithm as before using, but with the collection of data structures built for the dual representation Y', and following the steps of Theorem 2.2.

During the buildup of the recursive process, say being at step i, the size n'_i for sequence X'_i ($i > 1$), is upper bounded by $n/\log^{2i} m$, while the universe has size m/n^i. If at any step $2 \leq j \leq s$ the condition $m_j < w = n$ does not apply, we cannot apply Theorem 4.2, so we truncate recursion and use a $o(w)$ FID to store the sequence X_j. This contributes a negligible amount to the redundancy. We name the FID for each step F_1 up to F_s. Suppose we can recurse for s steps with Theorem 4.2, we end up with a sequence over a universe $m_s = m/n^s$. By using Corollary 3.2 the space bound is no less than $O(n(m/n^s)^{\varepsilon})$. The $B(n_i, m_i) + O(n_i^{1+\delta})$ factors decrease geometrically, so the root dominates and we can show that, apart from lower order terms, the space bound is as claimed. Otherwise, the total space $s(n_i, m_i)$ of the recursive data structure satisfies:

$$s(n_i, m_i) = s(n_{i+1}, m_{i+1}) + \text{space(FID for high bits)} + \text{space(string B-trees)} + O(n_i^{1+\delta})$$

where $n_{i+1} = n_i/\log^2 m$ and $m_{i+1} = m_i/n$. The claimed redundancy follows easily. ∎

Acknowledgements. The first two authors would like to thank Sebastiano Vigna for precious discussion. Thanks also go to the anonymous referees for useful comments. Work partially supported by the MAINSTREAM Italian MIUR Project.

References

[1] P. Beame and F.E. Fich. Optimal bounds for the predecessor problem and related problems. *J. Comput. Syst. Sci*, 65:38–72, 2002.

[2] D. K. Blandford and G. E. Blelloch. Compact dictionaries for variable-length keys and data with applications. *ACM Transactions on Algorithms*, 4(2):17:1–17:25, May 2008.

[3] A. Brodnik and J.I. Munro. Membership in constant time and almost-minimum space. *SIAM J. Computing*, 28:1627–1640, 1999.

[4] H. Buhrman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? *SIAM Journal on Computing*, 31(6):1723–1744, December 2002.

[5] D.R. Clark and J.I. Munro. Efficient suffix trees on secondary storage. In *Proc. 7th ACM-SIAM SODA*, pages 383–391, 1996.

[6] F. Claude and G. Navarro. Practical rank/select queries over arbitrary sequences. In *Proc. 15th (SPIRE)*, LNCS. Springer, 2008.

[7] P. Elias. Efficient storage and retrieval by content and address of static files. *J. Assoc. Comput. Mach.*, 21(2):246–260, 1974.

[8] R. M. Fano. On the number of bits required to implement an associative memory. *Memorandum 61, Computer Structures Group, Project MAC*, 1971.

[9] P. Ferragina, R. Grossi, A. Gupta, R. Shah, and J. S. Vitter. On searching compressed string collections cache-obliviously. In *Proc. 27th ACM PODS*, pages 181–190, 2008.
[10] P. Ferragina and Roberto Grossi. The string b-tree: A new data structure for string search in external memory and its applications. *J. of the ACM*, 46(2):236–280, 1999.
[11] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *Proc. 46th IEEE FOCS*, pages 184–196, 2005.
[12] P. Ferragina and G. Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, July 2005.
[13] A. Gál and P. Bro Miltersen. The cell probe complexity of succinct data structures. *Theor. Comput. Sci.*, 379:405–417, 2007.
[14] R. F. Geary, N. Rahman, R. Raman, and V. Raman. A simple optimal representation for balanced parentheses. *Theor. Comput. Sci.*, 368:231–246, 2006.
[15] R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2:510–534, 2006.
[16] A. Golynski. Optimal lower bounds for rank and select indexes. *Theor. Comput. Sci.*, 387:348–359, 2007.
[17] A. Golynski, R. Grossi, A. Gupta, R. Raman, and S. S. Rao. On the size of succinct indices. In *Proc 15th ESA, LNCS 4698*, pages 371–382, 2007.
[18] A. Golynski, R. Raman, and S. S. Rao. On the redundancy of succinct indices. In *Proc. 11th SWAT*, pages 148–159, 2008.
[19] R. González, Sz. Grabowski, V. Mäkinen, and G. Navarro. Practical implementation of rank and select queries. In *Proc. 4th (WEA)*, pages 27–38, 2005.
[20] R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th ACM-SIAM SODA*, pages 841–850, 2003.
[21] R. Grossi, A. Gupta, and J. S. Vitter. When indexing equals compression: experiments with compressing suffix arrays and applications. In *Proc. 15th ACM-SIAM SODA*, pages 636–645, 2004.
[22] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput*, 35(2):378–407, 2005.
[23] A. Gupta, W. Hon, R. Shah, and Jeffrey Scott Vitter. Compressed data structures: Dictionaries and data-aware measures. *Theor. Comput. Sci*, 387(3):313–331, 2007.
[24] W. Hon, K. Sadakane, and W. Sung. Breaking a time-and-space barrier in constructing full-text indices. In *Proc. 44th IEEE FOCS*, pages 251–260, 2003.
[25] G. Jacobson. *Succinct Static Data Structures*. PhD thesis, Carnegie Mellon University, 1989.
[26] P. B. Miltersen. Lower bounds on the size of selection and rank indexes. In *Proc. ACM-SIAM SODA*, pages 11–12, 2005.
[27] J. I. Munro. Tables. In *Proc. FST & TCS, LNCS 1180*, pages 37–42, 1996.
[28] J. I. Munro. Lower bounds for succinct data structures. In *Proc. 19th CPM*, page 3, 2008.
[29] J I. Munro, R. Raman, V. Raman, and S. S. Rao. Succinct representations of permutations. In *Proc. 30th ICALP, LNCS 2719*, pages 345–356, 2003.
[30] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31:762–776, 2001.
[31] J. I. Munro, V. Raman, and S. S. Rao. Space efficient suffix trees. *J. of Algorithms*, 39:205–222, 2001.
[32] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2:1–2:61, 2007.
[33] D. Okanohara and K. Sadakane. Practical entropy-compressed rank/select dictionary. In *ALENEX*. SIAM, 2007.
[34] R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Computing*, 31:353–363, 2001.
[35] M. Pătraşcu. Succincter. In *To Appear in Proc. 49th IEEE FOCS*, 2008.
[36] M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *Proc. 38th ACM STOC*, pages 232–240, 2006.
[37] R. Raman, V. Raman, and S. S. Rao. Succinct indexable dictionaries, with applications to representing k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4), 2007.
[38] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.
[39] S. Vigna. Broadword implementation of rank/select queries. In *Proc. 7th WEA*, pages 154–168, 2008.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 529–540
www.stacs-conf.org

A UNIFIED ALGORITHM FOR ACCELERATING EDIT-DISTANCE COMPUTATION VIA TEXT-COMPRESSION

DANNY HERMELIN[*] [1] AND GAD M. LANDAU[†] [1,2] AND SHIR LANDAU [3] AND OREN WEIMANN[‡] [4]

[1] Department of Computer Science, University of Haifa, Haifa, Israel.
E-mail address: danny@cri.haifa.ac.il,landau@cs.haifa.ac.il

[2] Department of Computer and Information Science, Polytechnic Institute of NYU, NY, USA.

[3] Department of Computer Science,, Bar-Ilan University, Ramat Gan, Israel.
E-mail address: shir.landau@live.biu.ac.il

[4] MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA.
E-mail address: oweimann@mit.edu

Abstract. The edit distance problem is a classical fundamental problem in computer science in general, and in combinatorial pattern matching in particular. The standard dynamic-programming solution for this problem computes the edit-distance between a pair of strings of total length $O(N)$ in $O(N^2)$ time. To this date, this quadratic upper-bound has never been substantially improved for general strings. However, there are known techniques for breaking this bound in case the strings are known to compress well under a particular compression scheme. The basic idea is to first compress the strings, and then to compute the edit distance between the compressed strings.

As it turns out, practically all known $o(N^2)$ edit-distance algorithms work, in some sense, under the same paradigm described above. It is therefore natural to ask whether there is a single edit-distance algorithm that works for strings which are compressed under any compression scheme. A rephrasing of this question is to ask whether a single algorithm can exploit the compressibility properties of strings under any compression method, even if each string is compressed using a different compression. In this paper we set out to answer this question by using *straight-line programs*. These provide a generic platform for representing many popular compression schemes including the LZ-family, Run-Length Encoding, Byte-Pair Encoding, and dictionary methods.

For two strings of total length N having straight-line program representations of total size n, we present an algorithm running in $O(n^{1.4}N^{1.2})$ time for computing the edit-distance of these two strings under any rational scoring function, and an $O(n^{1.34}N^{1.34})$-time algorithm for arbitrary scoring functions. This improves on a recent algorithm of Tiskin that runs in $O(nN^{1.5})$ time, and works only for rational scoring functions.

Key words and phrases: edit distance, straight-line programs, dynamic programming acceleration via compression, combinatorial pattern matching.

[*] Supported by the Adams Fellowship of the Israel Academy of Sciences and Humanities.

[†] Partially supported by the Israel Science Foundation grant 35/05 and the Israel-Korea Scientific Research Cooperation.

‡ Partially supported by the Israel-Korea Scientific Research Cooperation and by the Center for Massive Data Algorithmics (MADALGO) – a center of the Danish National Research Foundation.

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 561-572
http://drops.dagstuhl.de/opus/volltexte/2009/1804

1. Introduction

The *edit distance* between two strings over a fixed alphabet Σ is the minimum cost of transforming one string into the other via a sequence of character deletion, insertion, and replacement operations [31]. The cost of these elementary editing operations is given by some scoring function which induces a metric on strings over Σ. The simplest and most common scoring function is the Levenshtein distance [16] which assigns a uniform score of 1 for every operation. Determining the edit-distance between a pair of strings is a fundamental problem in computer science in general, and in combinatorial pattern matching in particular, with applications ranging from database indexing and word processing, to bioinformatics [11].

The standard dynamic programming solution for computing the edit distance between a pair of strings $A = a_1a_2 \cdots a_N$ and $B = b_1b_2 \cdots b_N$ involves filling in an $(N+1)\times(N+1)$ table T, with $T[i, j]$ storing the edit distance between $a_1a_2 \cdots a_i$ and $b_1b_2 \cdots b_j$. The computation is done according to the base-case rules given by $T[0, 0] = 0$, $T[i, 0] = T[i-1, 0]+$ the cost of deleting a_i, and $T[0, j] = T[0, j-1]+$ the cost of inserting b_j, and according to the following dynamic programming step:

$$T[i,j] = \min \begin{cases} T[i-1,j] + \text{the cost of deleting } a_i \\ T[i,j-1] + \text{the cost of inserting } b_j \\ T[i-1,j-1] + \text{the cost of replacing } a_i \text{ with } b_j \end{cases} \tag{1.1}$$

Note that as T has $(N+1)^2$ entries, the time-complexity of the algorithm above is $O(N^2)$.

Compression is traditionally used to efficiently store data. In this paper, we focus on using compression to accelerate the dynamic-programming solution for the edit-distance problem described above. The basic idea is to first compress the strings, and then compute the edit distance between the compressed strings. Note that the "acceleration via compression" approach has been successfully applied also to other classical problems on strings. Various compression schemes, such as LZ77 [33], LZW-LZ78 [32], Huffman coding, Byte-Pair Encoding (BPE) [27], Run-Length Encoding (RLE), were employed to accelerate exact string matching [3, 13, 17, 20, 28], subsequence matching [9], approximate pattern matching [2, 12, 13, 24], and more [23].

Regarding edit-distance computation, Bunke and Csirik presented a simple algorithm for computing the edit-distance of strings that compress well under RLE [8]. This algorithm was later improved in a sequence of papers [5, 6, 10, 19] to an algorithm running in time $O(nN)$, for strings of total length N that encode into run-length strings of total length n. In [10], an algorithm with the same time complexity was given for strings that are compressed under LZW-LZ78, where n again is the length of the compressed strings. Note that this algorithm is also $O(N^2/\lg N)$ in the worst-case for any strings over constant-size alphabets.

The first paper to break the quadratic time-barrier of edit-distance computation was the seminal paper of Masek and Paterson [21], who applied the "Four-Russians technique" to obtain a running-time of $O(N^2/\lg N)$ for any pair of strings, and of $O(N^2/\lg^2 N)$ assuming a unit-cost RAM model. Their algorithm essentially exploits repetitions in the strings to obtain the speed-up, and so in many ways it can also be viewed as compression-based. In fact, one can say that their algorithm works on the "naive compression" that all strings over constant-sized alphabets have. A drawback of the the Masek and Paterson algorithm is that it can only be applied when the given scoring function is rational. That is, when all costs of

editing operations are rational numbers. Note that this restriction is indeed a limitation in biological applications, where PAM and evolutionary distance similarity matrices are used for scoring [10, 21]. For this reason, the algorithm in [10] mentioned above was designed specifically to work for arbitrary scoring functions. We mentioned also Bille and Farach-Colton [7] who extend the Masek and Paterson algorithm to general alphabets.

There are two important things to observe from the above: First, all known techniques for improving on the $O(N^2)$ time bound of edit-distance computation, essentially apply acceleration via compression. Second, apart from RLE, LZW-LZ78, and the naive compression of the Four-Russians technique, we do not know how to efficiently compute edit-distance under other compression schemes. For example, no algorithm is known which substantially improves $O(N^2)$ on strings which compress well under LZ77. Such an algorithm would be interesting since there are various types of strings that compress much better under LZ77 than under RLE or LZW-LZ78. In light of this, and due to the practical and theoretical importance of substantially improving on the quadratic lower bound of string edit-distance computation, we set out to answer the following question:

> "Is there a general compression-based edit-distance algorithm that can exploit the compressibility of two strings under *any* compression scheme?"

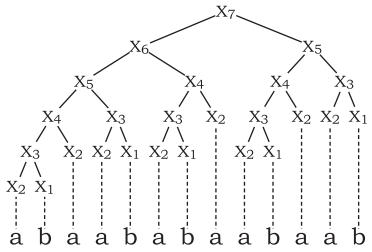
A key ingredient to answering this question, we believe, lies in a notion borrowed from the world of formal languages: The notion of straight-line programs.

1.1. Straight-line programs

A *straight-line program* (SLP) is a context-free grammar generating exactly one string. Moreover, only two types of productions are allowed: $X_i \to a$ where a is a unique terminal, and $X_i \to X_p X_q$ with $i > p, q$ where $X_1, \ldots, X_n$ are the grammar variables. Each variable appears exactly once on the left hand side of a production. The string represented by a given SLP is a unique string corresponding to the last nonterminal X_n. We define the size of an SLP to be n, the number of variables (or productions) it has. The length of the strings that is generated by the SLP is denoted by N. It is important to observe that many SLPs can be exponentially smaller than the string they generate.

Example 1.1. Consider the string *abaababaabaab*. It could be generated by the following SLP, also known as the *Fibonacci SLP*:

$$
\begin{aligned}
X_1 &\to b\\
X_2 &\to a\\
X_3 &\to X_2X_1\\
X_4 &\to X_3X_2\\
X_5 &\to X_4X_3\\
X_6 &\to X_5X_4\\
X_7 &\to X_6X_5
\end{aligned}
$$



Rytter [25] proved that the resulting encoding of most compression schemes including the LZ-family, RLE, Byte-Pair Encoding, and dictionary methods, can be transformed to straight-line programs quickly and without large expansion[1]. In particular, consider an

[1]Important exceptions of this list are statistical compressors such as Huffman or arithmetic coding, as well as compressions that are applied after a Burrows-Wheeler transformation.

LZ77 encoding [33] with n' blocks for a string of length N. Rytter's algorithm produces an SLP-representation with size $n = O(n' \log N)$ of the same string, in $O(n)$ time. Moreover, n lies within a $\log N$ factor from the size of a *minimal* SLP describing the same string. This gives us an efficient logarithmic approximation of minimal SLPs, since computing the LZ77 encoding of a string can be done in linear-time. Note also that any string compressed by the LZ78-LZW encoding can be transformed directly into a straight-line program within a constant factor.

1.2. Our results

Due to Rytter's results, SLPs are perfect candidates for achieving our goal of generalizing compression-based edit-distance algorithms. Indeed, a fast edit-distance algorithm for strings that have small SLP representations, would give a fast algorithm for strings which compress well under the compression schemes generalized by SLPs. Note that since constructing the strings generated by the SLPs requires linear-time in the length of the strings, an $O(N^2)$ algorithm is available via the standard dynamic-programming formulation (1.1). The main result of this paper gives an algorithm which beats this bound:

Theorem 1.2. *Let $\mathcal{A}$ and $\mathcal{B}$ be two SLPs of total size n that respectively generate two string A and B of total size N. Then, given $\mathcal{A}$ and $\mathcal{B}$, one can compute the edit-distance between A and B in $O(n^{1.4}N^{1.2})$ time for any rational scoring function.*

We can remove the dependency of rational scoring schemes in Theorem 1.2, recalling that arbitrary scoring schemes are important for biological applications. We obtain the following secondary result for arbitrary scoring functions:

Theorem 1.3. *Let $\mathcal{A}$ and $\mathcal{B}$ be two SLPs of total size n that respectively generate two string A and B of total size N. Then, given $\mathcal{A}$ and $\mathcal{B}$, one can compute the edit-distance between A and B in $O(n^{1.34}N^{1.34})$ time for any arbitrary scoring function.*

In the last part of the paper, we explain how the four-russians technique can also be incorporated into our SLP edit-distance scheme. We obtain a very simple algorithm that matches the performance of [10] in the worst-case. That is, we obtain a four-russian like algorithm with an $\Omega(\lg N)$ speed-up which can handle arbitrary scoring functions, unlike the Masek and Paterson algorithm which works only for rational functions. We add this algorithm to our presentation not only for its practical importance, but also to emphasize the fact that SLPs provide a framework which allows an almost perfect generalization of compression-based edit-distance algorithms.

1.3. Related Work

Rytter *et al.* [14] was the first to consider SLPs in the context of pattern matching, and other subsequent papers also followed this line [15, 22]. In [25] and [17] Rytter and Lifshits took this work one step further by proposing SLPs as a general framework for dealing with pattern matching algorithms that are accelerated via compression. However, the focus of Lifshits was on determining whether or not these problems are polynomial in n or not. In particular, he gave an $O(n^3)$-time algorithm to determine equality of SLPs [17], and he established hardness for the edit distance [18], and even for the hamming distance problems [17]. Nevertheless, Lifshits posed as an open problem the question of whether or not there is an $O(nN)$ edit-distance algorithm for SLPs. Here, our focus is on algorithms

which break the quadratic $O(N^2)$ time-barrier, and therefore all algorithms with running-times between $O(nN)$ and $O(N^2)$ are interesting for us.

Recently, Tiskin [29] gave an $O(nN^{1.5})$ algorithm for computing the longest common subsequence between two SLPs, an algorithm which can be extended at constant-factor cost to compute the edit-distance between the SLPs under any rational scoring function. Observe that our algorithm for arbitrary scoring functions in Theorem 1.3 is already faster than Tiskin's algorithm for most values of N and n. Also, it has the advantage of being much more simpler to implement. As for our main algorithm of Theorem 1.2, our faster running-time is achieved also by utilizing some of the techniques used by Tiskin in a more elaborate way.

2. The $DIST$ Table

The central dynamic-programming tool we use in our algorithms is the $DIST$ table, a simple and handy data-structure which was originally introduced by Apostolico *et al.* [4], and then further developed by others in [10, 26]. In the following section we briefly review basic facts about this tool that are essential for understanding our results, following mostly the presentation in [10]. We begin with the so-called dynamic-programming grid, a graph representation of edit-distance computation on which $DIST$ tables are defined.

Consider the standard dynamic programming formulation (1.1) for computing the edit-distance between two strings $A = a_1 a_2 \cdots a_N$ and $B = b_1 b_2 \cdots b_N$. The *dynamic-programming grid* associated with this program, is an acyclic-directed graph which has a vertex for each entry of T (see Figure 1). The vertex corresponding to $T[i, j]$ is associated with a_i and b_j, and has incoming edges according to (1.1) – an edge from $T[i-1, j]$ whose weight is the cost of deleting a_i, an edge from $T[i, j-1]$ whose weight is the cost of inserting b_j, and an edge from $T[i-1, j-1]$ whose weight is the cost of replacing a_i with b_j. The *value* at the vertex corresponding to $T[i, j]$ is the value stored in $T[i, j]$, *i.e.* the edit-distance between the length i prefix of A and the length j prefix of B. Using the dynamic-programming grid G, we reduce the problem of computing the edit-distance between A and B to the problem of computing the weight of the lightest path from the upper-left corner to bottom-right corner in G.

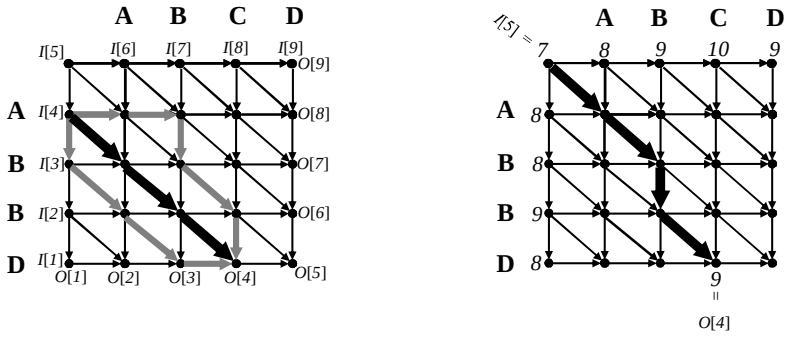


Figure 1: A subgraph of a Levenshtein distance dynamic program graph. On the left, $DIST[4, 4]$ (in bold) gives the minimum-weight path from $I[4]$ to $O[4]$. On the right, the value 9 of $O[4]$ is computed by $min_i I[i] + DIST[i, 4]$.

We will work with sub-grids of the dynamic-programming grid that will be referred to as *blocks*. The *input vertices* of a block are all vertices in the first row and column of the block, while its *output vertices* are all vertices in the last row and column. Together, the input and output vertices are referred to as the *boundary* of the block. The substrings of A and B associated with the block are defined in the straightforward manner according to its first row and column. Also, for convenience purposes, we will order the input and output vertices, with both orderings starting from the vertex in bottom-leftmost corner of the block, and ending at the vertex in the upper-rightmost corner. The ith input vertex and jth output vertex are the ith and jth vertices in these orderings. We next give the definition of $DIST$ tables, defined over blocks of G.

Definition 2.1 ($DIST$ [4])**.** Let G' be a block in G with x input vertices and x output vertices. The *DIST table* corresponding to G' is an $x \times x$ matrix, with $DIST[i, j]$ storing the weight of the minimum-weight path from the ith input to the jth output in G, and otherwise ∞ if no such paths exists.

It is important to notice that the values at the output vertices of a block are completely determined by that values at its input and its corresponding $DIST$ table. In particular, if $I[i]$ and $O[j]$ are the values at the ith input vertex and jth output vertex of a block G' of G, then

$$O[j] = \min_{1 \le i \le x} I[i] + DIST[i, j]. \tag{2.1}$$

Equation 2.1 implies not only the input-output relation of the dynamic-programming values of a block, but also that the values at the output vertices can be computed in linear time from the values at the input vertices. Indeed, by (2.1), the values at the output vertices of G' are given by the column minima of the matrix $I+DIST$. Furthermore, by a simple modification of all ∞ values in $I+DIST$, we get what is known as a *totally-monotone matrix* [10]. Now, Aggarwal *et al.* [1] gave a simple recursive algorithm, nicknamed SMAWK in the literature, that computes all column minima of an $x \times x$ totally-monotone matrix by querying only $O(x)$ elements of the matrix. It follows that using SMAWK we can compute the output values of G' in $O(x)$ time.

Let us now discuss how to efficiently construct the $DIST$ table corresponding to a block in G. Observe that this can be done quite easily in $O(x^3)$ time, for blocks with boundary size $O(x)$, by computing the standard dynamic-programming table between every prefix of A against B and every prefix of B against A. Each of these dynamic-programming tables contains all values of a particular row in the $DIST$ table. In [4], Apostolico *et al.* show an elegant way to reduce the time-complexity of this construction to $O(x^2 \lg x)$. In the case of rational scoring functions, the complexity can be further reduced to $O(x^2)$ as shown by Schmidt [26].

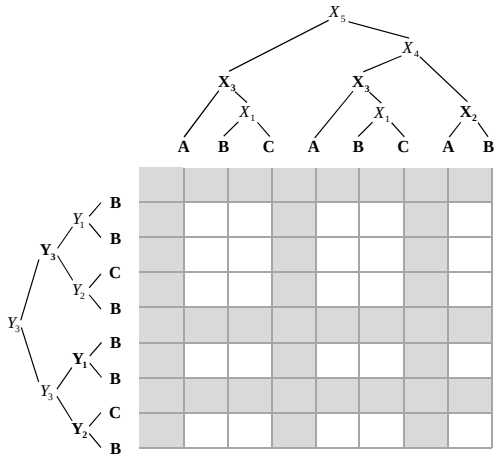
3. Acceleration via Straight-Line Programs

In the following section we describe a generic framework for accelerating the edit distance computation of two strings which are given by their SLP representation. This framework will later be used for explaining all our algorithms. We will refer throughout the paper to this framework as the *block edit-distance* procedure.

Let $\mathcal{A}$ and $\mathcal{B}$ be two SLP representations of a pair of strings A and B, and for ease of presentation assume that $|\mathcal{A}| = |\mathcal{B}| = n$ and $|A| = |B| = N$. Recall the definition in

Section 2 for the dynamic-programming grid corresponding to A and B. The general idea behind the block edit-distance procedure is to partition this grid into disjoint blocks, and then to compute the edit-distance between A and B at the cost of computing the values at the boundary vertices of each block. This is achieved by building in advance a repository containing all $DIST$ tables corresponding to blocks in the partition. To efficiently construct this repository, we show how to partition the grid in a way which induces many block repeats. This is possible by utilizing substring repeats in A and B that are captured in $\mathcal{A}$ and $\mathcal{B}$, and imply block repeats in the partitioning of G. The edit-distance of A and B is then computed by propagating the dynamic programming values at the boundary vertices of the blocks using the $DIST$ tables in the repository and SMAWK. Before giving a complete description of this algorithm, we need to introduce the notion of xy-partition.

Definition 3.1 (xy-partition)**.** An xy-partition is a partitioning of G into disjoint blocks such that every block has boundary of size $O(x)$, and there are $O(y)$ blocks in each row and column. In addition, we require each pair of substrings of A and B associated with a block to be generated by a pair of SLP variables in $\mathcal{A}$ and $\mathcal{B}$.



An xy-partition of an edit distance graph for two SLPs generating the strings "ABCABCAB" and "BBCBBBCB". The white blocks are the ones of the partition and their corresponding SLP variables are marked in bold. Notice that there are nine blocks in the partition but only six of them are distinct.

Figure 2: An xy-partition.

An xy-partition of G is a partition with a specific structure, but more importantly, one where each substring is generated by a unique SLP variable of $\mathcal{A}$ and $\mathcal{B}$. This latter requirement allows us to exploit the repetitions of A and B captured by their SLPs. We next give a complete description of the block edit distance procedure. It assumes an xy-partition of G has already been constructed. Section 4 explains how to construct such partitions.

Block Edit Distance

(1) Construct a repository with the $DIST$ tables corresponding to each block in the xy-partition.
(2) Fill-in the first row and column of G using the standard base-case rules.
(3) In top-to-bottom and left-to-right manner, identify the next block in the partition of G and use its input and the repository to compute its output using (2.1).
(4) Use the outputs in order to compute the inputs of the next blocks using (1.1).
(5) The value in the bottom-rightmost cell is the edit distance of A and B.

Apart from the repository construction in step 1, all details necessary for implementing the block edit-distance procedure are by now clear. Indeed, steps 2 and 5 are trivial, and step 4 is done via the standard dynamic-programming formulation of (1.1). Furthermore, the SMAWK computation of output values of a block, given its input values plus its corresponding $DIST$ table (step 3), is explained in Section 2. We next show that, as we are working with xy-partitions where each block is associated with an SLP variable, we can compute a repository containing all $DIST$ necessary which is rather small.

The first crucial observation for this, is that any two blocks associated with the same pair of substrings A' and B' have the same $DIST$ table. This is immediate since any such pair of blocks have identical edge-weights.

Observation 3.2. A pair of substrings A', B' uniquely identify the $DIST$ table of a block.

Since we required each substring in the xy-partition of G to be generated by some SLP variable, the above observation actually suggests that the number of different $DIST$ tables is bounded by the number of variable pairs $X \in \mathcal{A}$ and $Y \in \mathcal{B}$:

Observation 3.3. The number of different $DIST$ tables corresponding to any xy-partition is $O(n^2)$.

Therefore, combining the two observations above, we know that a repository containing a $DIST$ tables for each SLP variable pair $X \in \mathcal{A}$ and $Y \in \mathcal{B}$ will not be too large, and that it will contain a table corresponding to each block in our given xy-partition at hand. We can therefore state the following lemma:

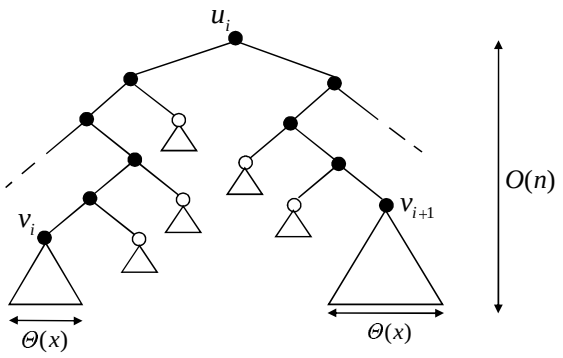
Lemma 3.4. *The block edit-distance procedure runs in $O(n^2x^2 \lg x + Ny)$ time.*

Proof. We analyze the time complexity of each step in the block edit-distance procedure separately. Step 1 can be performed in $O(n^2x^2 \lg x)$ time, as we can construct every $DIST$ table in $O(x^2 \lg x)$ time (see Section 2), and the total number of such distinct matrices is $O(n^2)$. Step 2 can be done trivially in $O(N)$ time. Then, step 3 takes $O(x)$ time per block by using the SMAWK algorithm as explained in Section 2. Step 4 also takes $O(x)$ time per block as it only computes the values in the $O(x)$ vertices adjacent to the output vertices. The total time complexity of steps 3 and 4 is thus equal to the total number of boundary vertices in the xy-partition of G, and therefore to $O(Ny)$. Accounting for all steps together, this gives us the time complexity stated in the lemma. ∎

4. Constructing an xy-partition

In this section we discuss the missing component of Section 3, namely the construction of xy-partitions. In particular, we complete the proof of Theorem 1.3 by showing how to efficiently construct an xy-partition where $y = O(nN/x)$ for every $x \leq N$. Together with Lemma 3.4, this implies an $O(n^{\frac{4}{3}}N^{\frac{4}{3}} \lg^{\frac{1}{3}} N) = O(n^{1.34}N^{1.34})$ time algorithm for arbitrary scoring functions by considering $x = N^{\frac{2}{3}}/(n \lg N)^{\frac{1}{3}}$. In the remainder of this section we prove the following lemma.

Lemma 4.1. *For every $x \leq N$ there exists an xy-partition with $y = O(nN/x)$. Moreover, this partition can be found in $O(N)$ time.*



The key-vertices v_i and v_{i+1} both generate substrings of length $\Theta(x)$, and their least common ancestor is u_i. The white vertices "hanging" of the v_i-to-v_{i+1} path are the added key-vertices. Together, they generate the substring that lies between the substrings of v_i and v_{i+1}.

Figure 3: A closer look on the parse tree of an SLP $\mathcal{A}$.

To prove the lemma, we show that for every SLP $\mathcal{A}$ generating a string A and every $x \leq N$, one can partition A into $O(nN/x)$ disjoint substrings, each of length $O(x)$, such that every substring is generated by some variable in $\mathcal{A}$. This defines a subset of variables in both input SLPs which together defined our desired xy-partition. To partition A, we first identify $O(N/x)$ grammar variables in $\mathcal{A}$ each generating a disjoint substring of length between x and $2x$. We use these variables to partition A. We then show that the substrings of A that are still not associated with a variable can each be generated by $O(n)$ additional variables. Furthermore, these $O(n)$ variables each generate a string of length bounded by x. We add all such variables to our partition of A for a total of $O(nN/x)$ variables.

Consider the parse tree of $\mathcal{A}$. We want to identify $O(nN/x)$ *key-vertices* such that every key-vertex generates a substring of length $O(x)$, and A is a concatenation of substrings generated by key-vertices. We start by marking every vertex v that generates a substring of length greater than x as a key-vertex iff both children of v generate substrings of length smaller than x. This gives us $\ell \leq N/x$ key-vertices so far, each generating a substring of length $\Theta(x)$ (see Figure 3). But we are still not guaranteed that these vertices cover A entirely.

To fix this, consider the ordering $v_1, v_2, \ldots, v_\ell$ on the current key-vertices induced by a left-to-right postorder traversal of the parse tree. This way, v_{i+1} is "to the right of" v_i. If every v_i generates the substring A_i then $A = A'_1 A_1 A'_2 A_2 \cdots A'_\ell A_\ell A'_{\ell+1}$, where every A_i is of length $\Theta(x)$, and every A'_i is the "missing" substring of A that lies between A_{i-1} and A_i. We now show that every A'_i is a concatenation of substrings of length smaller than x generated by at most $O(n)$ vertices.

Let u_i be the lowest common ancestor of v_i and v_{i+1} and let P_i (resp. P_{i+1}) be the unique path between u_i and v_i (resp. v_{i+1}). For every vertex $v \in P_i - \{u_i\}$ such that v's left child is also in P_i mark v's right child as a key-vertex. Similarly, for every vertex $v \in P_{i+1} - \{u_i\}$ such that v's right child is also in P_{i+1} mark v's left child as a key-vertex. It is easy to verify that A'_i is the concatenation of substrings generated by these newly marked key-vertices. There are at most $2n$ of these key-vertices since the depth of the parse tree is bounded by the number of different SLP variables. Moreover, they each generate a substring of length smaller than x for the following reason. Assume for contradiction that one of them generates a string of length greater than x. This would imply the existence of some vertex between v_i and v_{i+1} in the $v_1, v_2, \ldots, v_\ell$ ordering.

To conclude, we showed that $A = A'_1 A_1 A'_2 A_2 \cdots A'_\ell A_\ell A'_{\ell+1}$ where $\ell \leq N/x$, every A_i is of length $\Theta(x)$ and is generated by one vertex, and every A'_i is a concatenation of $O(n)$ substrings each of length smaller than x and generated by one vertex. Overall, we get that $y = O(n\ell) = O(nN/x)$ vertices suffice to generate A for every $x \leq N$. It is easy to see that we can identify these vertices in $O(N)$ time thus proving Lemma 4.1. By choosing $x = N^{\frac{2}{3}}/(n \lg N)^{\frac{1}{3}}$, and using the block edit distance time complexity of Lemma 3.4, this implies an $O(n^{1.34}N^{1.34})$ time algorithm for arbitrary scoring functions.

5. Improvement for Rational Scoring Functions

In this section we show that in the case of rational scoring functions, the time complexity of the block edit distance procedure can be reduced substantially by using a recursive construction of the $DIST$ tables. In particular, we complete the proof of Theorem 1.2 by showing that in this case the repository of $DIST$ tables can be computed in $O(n^2x^{1.5})$ time. This implies an $O(n^{1.4}N^{1.2})$ time algorithm for rational scoring functions by considering $x = N^{0.8}/n^{0.4}$ and the xy-partition with $y = nN/x$.

Before we describe how to compute the repository in $O(n^2x^{1.5})$ time, we need to introduce some features that $DIST$ tables over rational scoring functions have. The first property, discovered by Schmidt [26], is what is known as the succinct representation property: Any $x \times x$ $DIST$ table can be succinctly stored using only $O(x)$ space. This follows from considering the vector obtained by subtracting a $DIST$ column from the column to its right, and observing that this vector has only a constant number of value changes. The second property is that succinct representations allow to efficiently merge two $DIST$ tables. That is, if D_1 and D_2 are two $DIST$ tables, one between a pair of substrings A' and B' and the other between A' and B'', then we refer to the $DIST$ table between A' and $B'B''$ as the product of *merging* D_1 and D_2. A recent important result of Tiskin [30] shows how to utilize the succinct representation of $DIST$ tables in order to merge two succinct $x \times x$ $DIST$ tables in $O(x^{1.5})$ time.

Lemma 5.1. *The block edit distance algorithm runs in $O(n^2x^{1.5} + Ny)$ time in case the underlying scoring function is rational.*

Proof. To prove the lemma it suffices to show how to compute the repository of $DIST$ tables in step 1 of the block edit-distance procedure in $O(n^2x^{1.5})$ time, in case the underlying scoring function is rational. We will work with succinct representations of the $DIST$ tables as described above. Say $X \to X_pX_q$ and $Y \to Y_sY_t$ are two rules in the SLPs $\mathcal{A}$ and $\mathcal{B}$ respectively. To compute the $DIST$ table that corresponds to the strings generated by X and Y, we first recursively compute the four $DIST$ tables that correspond to the pairs (X_p, Y_s), (X_p, Y_t), (X_q, Y_s), and (X_q, Y_t). We then merge these four tables to obtain the $DIST$ table that corresponds to (X, Y). To do so we use Tiskin's procedure to merge (X_p, Y_s) with (X_p, Y_t) into (X_p, Y_sT_t), then merge (X_q, Y_s) with (X_q, Y_t) into (X_q, Y_sT_t), and finally we merge (X_p, Y_sT_t) and (X_q, Y_sT_t) into $(X_pX_q, Y_sT_t) = (X, Y)$. This recursive procedure computes each succinct $DIST$ table by three merge operations, each taking $O(x^{1.5})$ time and $O(x)$ space. Since the number of different $DIST$ tables is bounded by $O(n^2)$, the $O(n^2x^{1.5})$ time for constructing the repository follows. ∎

To conclude, we have shown an $O(n^2x^{1.5} + Ny)$ time algorithm for computing the edit distance. Using the xy-partition from Lemma 4.1 with $x = N^{0.8}/n^{0.4}$ and $y = nN/x$, we get a time complexity of $O(n^{1.4}N^{1.2})$.

6. Four-Russian Interpretation

In the previous sections we showed how SLPs can be used to speed up the edit distance computation of strings that compress well under some compression scheme. In this section, we conclude the presentation of our SLP framework by presenting an $\Omega(\lg N)$ speed-up for strings that do not compress well under any compression scheme. To do so, we adopt the Four Russions approach of Masek and Paterson [21] that utilizes a naive property that every string over a fixed alphabet has. Namely, that short enough substrings must appear many times. However, while the Masek and Paterson algorithm can only handle rational scoring functions, the SLP version that we propose can handle arbitrary scoring functions.

Consider a string A of length N over an alphabet Σ. The parse tree of the naive SLP $\mathcal{A}$ is a complete binary tree with N leaves[2]. This way, for every $x \leq N$ we get that A is the concatenation of $O(N/x)$ substrings each of length $\Theta(x)$ and each can be generated by some variable in $\mathcal{A}$. This partition of A suggests an xy-partition in which $y = N/x$. At first glance, this might seem better than the partition guarantee of Lemma 4.1 in which $y = nN/x$. However, notice that in the naive SLP we have $n \geq N$ so we can not afford to compute a repository of $O(n^2)$ $DIST$ tables.

To overcome this problem, we choose x small enough so that $|\Sigma|^x$, the number of possible substrings of length x, is small. In particular, by taking $x = \frac{1}{2}\log_{|\Sigma|} N$ we get that the number of possible substrings of length x is bounded by $|\Sigma|^x = \sqrt{N}$. This implies an xy-partition in which $x = \frac{1}{2}\log_{|\Sigma|} N$, $y = N/x$, and the number of distinct blocks n' is $O(N)$. Using this partition, we get that the total construction time of the $DIST$ repository is $O(n'x^2 \lg x)$. Similar to Lemma 3.4, we get that the total running time of the block edit distance algorithm is $O(n'x^2 \lg x + Ny)$ which gives $O(N^2/\lg N)$.

References

[1] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[2] A. Amir, G. Benson, and M. Farach. Let sleeping files lie: Pattern matching in Z-compressed files. *Journal of Comp. and Sys. Sciences*, 52(2):299–307, 1996.

[3] A. Amir, G.M. Landau, and D. Sokol. Inplace 2d matching in compressed images. In *Proc. of the 14th annual ACM-SIAM Symposium On Discrete Algorithms, (SODA)*, pages 853–862, 2003.

[4] A. Apostolico, M.J. Atallah, L.L. Larmore, and S.McFaddin. Efficient parallel algorithms for string editing and related problems. *SIAM Journal on Computing*, 19(5):968–988, 1990.

[5] A. Apostolico, G.M. Landau, and S. Skiena. Matching for run length encoded strings. *Journal of Complexity*, 15(1):4–16, 1999.

[6] O. Arbell, G. M. Landau, and J. Mitchell. Edit distance of run-length encoded strings. *Information Processing Letters*, 83(6):307–314, 2001.

[7] P. Bille and M. Farach-Colton. Fast and compact regular expression matching. *CoRR*, 2005.

[8] H. Bunke and J. Csirik. An improved algorithm for computing the edit distance of run length coded strings. *Information Processing Letters*, 54:93–96, 1995.

[9] P. Cégielski, I. Guessarian, Y. Lifshits, and Y. Matiyasevich. Window subsequence problems for compressed texts. In *Proc. of the 1st symp. on Computer Science in Russia (CSR)*, pages 127–136, 2006.

[2]We assume without loss of generality that N is a power of 2.

[10] M. Crochemore, G.M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32:1654–1673, 2003.
[11] D. Gusfield. *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology.* Cambridge University Press, 1997.
[12] J. Karkkainen, G. Navarro, and E. Ukkonen. Approximate string matching over Ziv-Lempel compressed text. In *Proc. of the 11th symposium on Combinatorial Pattern Matching (CPM)*, pages 195–209, 2000.
[13] J. Karkkainen and E. Ukkonen. Lempel-Ziv parsing and sublinear-size index structures for string matching. In *Proc. of the 3rd South American Workshop on String Processing (WSP)*, pages 141–155, 1996.
[14] M. Karpinski, W. Rytter, and A. Shinohara. Pattern-matching for strings with short descriptions. In *Proc. of the 6th symposium on Combinatorial Pattern Matching (CPM)*, pages 205–214, 1995.
[15] E. Lehman and A. Shelat. Approximation algorithms for grammar-based compression. In *Proc. of the 13th annual ACM-SIAM Symposium On Discrete Algorithms, (SODA)*, pages 205–212, 2002.
[16] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
[17] Y. Lifshits. Processing compressed texts: A tractability border. In *Proc. of the 18th symposium on Combinatorial Pattern Matching (CPM)*, pages 228–240, 2007.
[18] Y. Lifshits and M. Lohrey. Querying and embedding compressed texts. In *Proc. of the 31st international symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 681–692, 2006.
[19] V. Makinen, G. Navarro, and E. Ukkonen. Approximate matching of run-length compressed strings. In *Proc. of the 12th Symposium On Combinatorial Pattern Matching (CPM)*, pages 1–13, 1999.
[20] U. Manber. A text compression scheme that allows fast searching directly in the compressed file. In *Proc of the 5th Symposium On Combinatorial Pattern Matching (CPM)*, pages 31–49, 1994.
[21] W.J. Masek and M.S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20, 1980.
[22] M. Miyazaki, A. Shinohara, and M. Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. In *Proc. of the 8th symposium on Combinatorial Pattern Matching (CPM)*, pages 1–11, 1997.
[23] S. Mozes, O. Weimann, and M. Ziv-Ukelson. Speeding up HMM decoding and training by exploiting sequence repetitions. In *Proc. of the 18th symposium on Combinatorial Pattern Matching (CPM)*, pages 4–15, 2007.
[24] G. Navarro, T. Kida, M. Takeda, A. Shinohara, and S. Arikawa. Faster approximate string matching over compressed text. In *Proc. of the 11th Data Compression Conference (DCC)*, pages 459–468, 2001.
[25] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1-3):211–222, 2003.
[26] J.P. Schmidt. All highest scoring paths in weighted grid graphs and their application to finding all approximate repeats in strings. *SIAM Journal on Computing*, 27(4):972–992, 1998.
[27] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Byte Pair encoding: A text compression scheme that accelerates pattern matching. *Technical Report DOI-TR-161, Department of Informatics, Kyushu University*, 1999.
[28] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Speeding up pattern matching by text compression. In *Proc. of the 4th Italian Conference Algorithms and Complexity (CIAC)*, pages 306–315, 2000.
[29] A. Tiskin. Faster subsequence recognition in compressed strings. *J. of Mathematical Sciences, to appear.*
[30] A. Tiskin. All semi-local longest common subsequences in subquadratic time. In *Proc. of the 1st Computer Science symposium in Russia (CSR)*, pages 352–363, 2006.
[31] R. Wagner and M. Fischer. The string-to-string correction problem. *J. of the ACM*, 21(1):168–173, 1974.
[32] J. Ziv and A. Lempel. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.
[33] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 541–552
www.stacs-conf.org

RANDOM FRUITS ON THE ZIELONKA TREE

FLORIAN HORN[1]

[1] CWI, Amsterdam, The Netherlands
E-mail address: f.horn@cwi.nl

Abstract. Stochastic games are a natural model for the synthesis of controllers confronted to adversarial and/or random actions. In particular, ω-regular games of infinite length can represent reactive systems which are not expected to reach a correct state, but rather to handle a continuous stream of events. One critical resource in such applications is the memory used by the controller. In this paper, we study the amount of memory that can be saved through the use of randomisation in strategies, and present matching upper and lower bounds for stochastic Muller games.

1. Introduction

A stochastic game arena is a directed graph with three kinds of states: Eve's, Adam's and random states. A token circulates on this arena: when it is in one of Eve's states, she chooses its next location among the successors of the current state; when it is in one of Adam's states, he chooses its next location; and when it is in a random state, the next location is chosen according to a fixed probability distribution. The result of playing the game for ω moves is an infinite path of the graph. A play is winning either for Eve or for Adam, and the "winner problem" consists in determining whether one of the players has a winning strategy, from a given initial state. Closely related problems concern the computation of winning strategies, as well as determining the nature of these strategies: pure or randomised, with finite or infinite memory. There has been a long history of using arenas without random states (2-player arenas) for modelling and synthesising reactive processes [BL69, PR89]: Eve represents the controller, and Adam the environment. Stochastic ($2\frac{1}{2}$-player) arenas [deA97], with the addition of random states, can also model uncontrollable actions that happen according to a random law, rather than by choice of an actively hostile environment. The desired behaviour of the system is traditionally represented as an ω-regular winning condition, which naturally expresses the temporal specifications and fairness assumptions of transition systems [MP92]. From this point of view, the complexity of the winning strategies is a central question, since they represent possible implementations of the controllers in the synthesis problem. In this paper, we focus on an important normal form of ω-regular conditions, namely *Muller* winning conditions (see [Tho95] for a survey).

1998 ACM Subject Classification: D.2.4. Model Checking (Theory).
Key words and phrases: model checking, controller synthesis, stochastic games, randomisation.
This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme.

In the case of 2-player Muller games, a fundamental determinacy result of Büchi and Landweber states that, from any initial state, one of the players has a winning strategy [BL69]. Gurevich and Harrington used the *latest appearance record* (LAR) structure of McNaughton to extend this result to strategies with memory factorial in the size of the game [GH82]. Zielonka refines the LAR construction into a tree, and derives from it an elegant algorithm to compute the winning regions [Zie98]. An insightful analysis of the Zielonka tree by Dziembowski, Jurdzinski, and Walukiewicz leads to optimal (and asymmetrical) memory bounds for pure (non-randomised) winning strategies [DJW97]. Chatterjee extended these bounds to the case of pure strategies over $2\frac{1}{2}$-player arenas [Cha07b]. However, the lower bound on memory does not hold for *randomised strategies*, even in non-stochastic arenas: Chatterjee, de Alfaro, and Henzinger show that memoryless randomised strategies are enough for to deal with upward-closed winning conditions [CdAH04]. Chatterjee extends this result in [Cha07a], showing that conditions with non-trivial upward-closed subsets admit randomised strategies with less memory than pure ones.

Our contributions. The memory bounds of [Cha07a] are not tight in general, even for 2-player arenas. We give here matching upper and lower bounds for any Muller condition $\mathcal{F}$, in the form of a number $r_{\mathcal{F}}$ computed from the Zielonka tree of $\mathcal{F}$:

- if Eve has a winning strategy in a $2\frac{1}{2}$-player game $(\mathcal{A}, \mathcal{F})$, she has a randomised winning strategy with memory $r_{\mathcal{F}}$ (Theorem 4.2);
- there is a 2-player game $(\mathcal{A}_{\mathcal{F}}, \mathcal{F})$ where any randomised winning strategy for Eve has at least $r_{\mathcal{F}}$ memory states (Theorem 5.2).

Furthermore, the witness arenas we build in the proof of Theorem 5.2 are significantly smaller than in [DJW97], even though the problem of *polynomial* arenas remains open.

Outline of the paper. Section 2 recalls the classical notions in the area, while Section 3 presents former results on memory bounds and randomised strategies. The next two sections present our main results. In Section 4, we introduce the number $r_{\mathcal{F}}$ and show that it is an upper bound on the memory needed to win in any $2\frac{1}{2}$-game $(\mathcal{A}, \mathcal{F})$. In Section 5, we show that this bound is tight. Finally, in Section 6, we characterise the class of Muller conditions that admit memoryless randomised strategies, and show that for each Muller condition, at least one of the players cannot improve its memory through randomisation.

2. Definitions

We consider turn-based stochastic two-player Muller games. We recall here several classical notions in the field, and refer the reader to [Tho95, deA97] for more details.

Probability Distribution. A probability distribution γ over a set X is a function from X to $[0,1]$ such that $\sum_{x \in X} \gamma(x) = 1$. The set of probability distributions over X is denoted by $\mathcal{D}(X)$.

Arenas. A *$2\frac{1}{2}$-player arena* $\mathcal{A}$ over a set of colours $\mathcal{C}$ consists of a directed finite graph $(\mathcal{S}, \mathcal{T})$, a partition $(\mathcal{S}_E, \mathcal{S}_A, \mathcal{S}_R)$ of $\mathcal{S}$, a probabilistic transition function $\delta : S_R \to \mathcal{D}(S)$ such that $\delta(s)(t) > 0 \Leftrightarrow (s,t) \in \mathcal{T}$, and a partial colouring function $\chi : \mathcal{S} \rightharpoonup \mathcal{C}$. The states in $\mathcal{S}_E$ (resp. $\mathcal{S}_A$, $\mathcal{S}_R$) are *Eve's states* (resp. *Adam's states*, *random states*), and are graphically represented as $\bigcirc$'s (resp. $\Box$, $\triangle$). A *2-player arena* is an arena where $\mathcal{S}_R = \emptyset$.

A set $U \subseteq \mathcal{S}$ of states is *δ-closed* if for every random state $u \in U \cap \mathcal{S}_R$, $(u,t) \in \mathcal{T} \to t \in U$. It is *live* if for every non-random state $u \in U \cap (\mathcal{S}_E \cup \mathcal{S}_A)$, there is a state $t \in U$ such that $(u,t) \in \mathcal{T}$. A live and δ-closed subset U induces a *subarena* of $\mathcal{A}$, denoted by $\mathcal{A} \restriction U$.

Plays and Strategies. An infinite path, or *play*, over the arena $\mathcal{A}$ is an infinite sequence $\rho = \rho_0\rho_1\dots$ of states such that $(\rho_i, \rho_{i+1}) \in \mathcal{T}$ for all $i \in \mathbb{N}$. The set of states occurring infinitely often in a play ρ is denoted by $\mathrm{Inf}(\rho) = \{s \mid \exists^\infty i \in \mathbb{N}, \rho_i = s\}$. We write Ω for the set of all plays, and Ω_s for the set of plays that start from the state s.

A *strategy with memory* M for Eve on the arena $\mathcal{A}$ is a (possibly infinite) transducer $\sigma = (M, \sigma^{\mathtt{n}}, \sigma^{\mathtt{u}})$, where $\sigma^{\mathtt{n}}$ is the "next-move" function from $(\mathcal{S}_E \times M)$ to $\mathcal{D}(\mathcal{S})$ and $\sigma^{\mathtt{u}}$ is the "memory-update" function, from $(\mathcal{S} \times M)$ to $\mathcal{D}(M)$. Notice that both the move and the update are randomised: strategies whose memory is deterministic are a different, less compact, model. The strategies for Adam are defined likewise. A strategy σ is *pure* if it does not use randomisation. It is *finite-memory* if M is a finite set, and *memoryless* if M is a singleton. Notice that strategies defined in the usual way as functions from $\mathcal{S}^*$ to $\mathcal{S}$ can be defined as strategies with infinite memory: the set of memory states is $\mathcal{S}^*$ and the memory update is $\sigma^{\mathtt{u}}(s, w) \mapsto ws$.

Once a starting state $s \in \mathcal{S}$ and strategies $\sigma \in \Sigma$ for both players are fixed, the outcome of the game is a random walk $\rho_s^{\sigma,\tau}$ for which the probabilities of events are uniquely fixed (an *event* is a measurable set of paths). For an event $P \in \Omega$, we denote by $\mathbb{P}_s^{\sigma,\tau}(P)$ the probability that a play belongs to P if it starts from s and Eve and Adam follow the strategies σ and τ.

A play is consistent with σ if for each position i such that $w_i \in \mathcal{S}_E$, $\mathbb{P}_{w_0}^{\sigma,\tau}(\rho_{i+1} = w_{i+1} \mid \rho_0 = w_0 \dots \rho_i = w_i) > 0$. The set of plays consistent with σ is denoted by Ω^σ. Similar notions can be defined for Adam's strategies.

Traps and Attractors. The *attractor of Eve to the set* U, denoted $\mathrm{Attr}_E(U)$, is the set of states where Eve can guarantee that the token reaches the set U with a positive probability. It is defined inductively by:

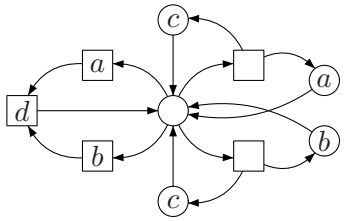
$$\begin{array}{rcll} \mathrm{Attr}_E^0(U) & = & U & \\ \mathrm{Attr}_E^{i+1}(U) & = & \mathrm{Attr}_E^i(U) & \cup\{s \in \mathcal{S}_E \cup \mathcal{S}_R, \exists t \in \mathrm{Attr}_E^i(U) \mid (s,t) \in \mathcal{T}\} \\ & & & \cup\{s \in \mathcal{S}_A \mid \forall t, (s,t) \in E \Rightarrow t \in \mathrm{Attr}_E^i(U)\} \\ \mathrm{Attr}_E(U) & = & \bigcup_{i>0} & \mathrm{Attr}_E^i(U) \end{array}$$

The corresponding *attractor strategy to U for Eve* is a pure and memoryless strategy a_U such that for any state $s \in \mathcal{S}_E \cap (\mathrm{Attr}_E(U) \setminus U)$, $s \in \mathrm{Attr}_E^{i+1}(U) \Rightarrow a_U(s) \in \mathrm{Attr}_E^i(U)$.

The dual notion of *trap for Eve* denotes a set from where Eve cannot escape, unless Adam allows her to do so: a set U is a trap for Eve if and only if $\forall s \in U \cap (\mathcal{S}_E \cup \mathcal{S}_R), (s,t) \in \mathcal{T} \Rightarrow t \in U$ and $\forall s \in U \cap \mathcal{S}_A, \exists t \in U, (s,t) \in \mathcal{T}$. Notice that a trap is a "strong" notion —the token can never leave it if Adam does not allow it to do so— while an attractor is a "weak" one —the token can avoid the target even if Eve uses the attractor strategy. Notice also that a trap (for either player) is always a subarena.

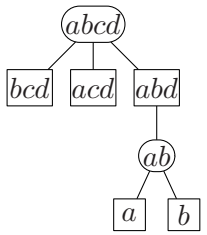
Winning Conditions. A *winning condition* is a subset Φ of Ω. A play ρ is *winning for Eve* if $\rho \in \Phi$, and *winning for Adam* otherwise. We consider ω-regular winning conditions formalised as *Muller conditions*. A Muller condition is determined by a subset $\mathcal{F}$ of the power set $\mathcal{P}(\mathcal{C})$ of colours, and Eve wins a play if and only if the set of colours visited infinitely often belongs to $\mathcal{F}$: $\Phi_\mathcal{F} = \{\rho \in \Omega | \chi(\mathrm{Inf}(\rho)) \in \mathcal{F}\}$. An example of Muller game is given in Figure 1(a). We use it throughout the paper to describe various notions and results.

Winning Strategies. A strategy σ for Eve is *surely winning* (or *sure)* from a state s for the winning condition Φ if any play consistent with σ belongs to Φ, and *almost-surely winning* (or *almost-sure*) if for any strategy τ for Adam, $\mathbb{P}_s^{\sigma,\tau}(\Phi) = 1$. The *sure* and *almost-sure* regions are the sets of states from which she has a sure (resp. almost-sure) strategy.



$\mathfrak{F} = \{\{a,b\},\{a,b,c\},\{a,b,c,d\}\}$

(a) The game $\mathfrak{G} = (\mathfrak{A}, \mathfrak{F})$



(b) Zielonka Tree of $\mathfrak{F}$

Figure 1: Recurring Example

3. Former results in memory bounds and randomisation

3.1. Pure strategies

There has been intense research since the sixties on the non-stochastic setting, *i.e.* pure strategies and 2-player arenas. Büchi and Landweber showed the determinacy of Muller games in [BL69]. Gurevich and Harrington used the LAR (Latest Appearance Record) of McNaughton to prove their *Forgetful Determinacy* theorem [GH82], which shows that a memory of size $|\mathcal{C}|!$ is sufficient for any game that uses only colours from $\mathcal{C}$, even when the arena is infinite. This result was later refined by Zielonka in [Zie98], using a representation of the Muller conditions as trees:

Definition 3.1 (Zielonka Tree of a Muller condition)**.** The Zielonka Tree $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$ of a winning condition $\mathcal{F} \subseteq \mathcal{P}(\mathcal{C})$ is defined inductively as follows:

(1) If $\mathcal{C} \notin \mathcal{F}$, then $\mathcal{Z}_{\mathcal{F},\mathcal{C}} = \mathcal{Z}_{\overline{\mathcal{F}},\mathcal{C}}$, where $\overline{\mathcal{F}} = \mathcal{P}(\mathcal{C}) \setminus \mathcal{F}$.

(2) If $\mathcal{C} \in \mathcal{F}$, then the root of $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$ is labelled with $\mathcal{C}$. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$ be all the maximal sets in $\{U \notin \mathcal{F} \mid U \subseteq \mathcal{C}\}$. Then we attach to the root, as its subtrees, the Zielonka trees of $\mathcal{F} \restriction \mathcal{C}_i$, *i.e.* the $\mathcal{Z}_{\mathcal{F}\restriction\mathcal{C}_i,\mathcal{C}_i}$, for $i = 1 \ldots k$.

Hence, the Zielonka tree is a tree with nodes labelled by sets of colours. A node of $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$ is an Eve node if it is labelled with a set from $\mathcal{F}$, otherwise it is an Adam node.

A later analysis of this construction by Dziembowski, Jurdzinski and Walukiewicz in [DJW97] led to an optimal and asymmetrical bound on the memory needed by the players to define sure strategies:

Definition 3.2 (Number $m_{\mathcal{F}}$ of a Muller condition)**.** Let $\mathcal{F} \subseteq \mathcal{P}(\mathcal{C})$ be a Muller condition, and $\mathcal{Z}_{\mathcal{F}_1,\mathcal{C}_1}, \mathcal{Z}_{\mathcal{F}_2,\mathcal{C}_2}, \ldots, \mathcal{Z}_{\mathcal{F}_k,\mathcal{C}_k}$ be the subtrees attached to the root of the tree $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$. We define the number $m_{\mathcal{F}}$ inductively as follows:

$$m_{\mathcal{F}} = \begin{cases} 1 & \text{if } \mathcal{Z}_{\mathcal{F},\mathcal{C}} \text{ does not have any subtrees,} \\ \max\{m_{\mathcal{F}_1}, m_{\mathcal{F}_2}, \ldots, m_{\mathcal{F}_k}\} & \text{if } \mathcal{C} \notin \mathcal{F} \text{ (Adam node),} \\ \sum_{i=1}^{k} m_{\mathcal{F}_i} & \text{if } \mathcal{C} \in \mathcal{F} \text{ (Eve node).} \end{cases}$$

Theorem 3.3 ([DJW97])**.** *If Eve has a sure strategy in a 2-player Muller game with the winning condition $\mathcal{F}$, she has a pure sure strategy with at most $m_{\mathcal{F}}$ memory states. Furthermore, there is a 2-player arena $\mathcal{A}_{\mathcal{F}}$ such that Eve has a sure strategy, but none of her sure strategies have less than $m_{\mathcal{F}}$ memory states.* ∎

Theorem 3.4 ([Cha07b]). *If Eve has an almost-sure strategy in a $2\frac{1}{2}$-player Muller game with the winning condition $\mathcal{F}$, she has a pure almost-sure with at most $m_{\mathcal{F}}$ memory states.* ∎

3.2. Memory reduction through randomisation

Randomised strategies are more general than pure strategies, and in some cases, they are also more compact. In [CdAH04], a first result showed that upward-closed conditions admit memoryless randomised strategies, while they don't admit memoryless pure strategies:

Theorem 3.5 ([CdAH04]). *If Eve has an almost-sure strategy in a $2\frac{1}{2}$-player Muller game with an upward-closed winning condition, she has a randomised almost-sure strategy.* ∎

This result was later extended in [Cha07a], by removing the leaves attached to a node of the Zielonka Tree representing an upward-closed subcondition:

Definition 3.6 ([Cha07a]). Let $\mathcal{F} \subseteq \mathcal{P}(\mathcal{C})$ be a Muller condition, and $\mathcal{Z}_{\mathcal{F}_1,\mathcal{C}_1}, \mathcal{Z}_{\mathcal{F}_2,\mathcal{C}_2}, \ldots, \mathcal{Z}_{\mathcal{F}_k,\mathcal{C}_k}$ be the subtrees attached to the root of the tree $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$. We define the number $m^U_{\mathcal{F}}$ inductively as follows:

$$m^U_{\mathcal{F}} = \begin{cases} 1 & \text{if } \mathcal{Z}_{\mathcal{F},\mathcal{C}} \text{ does not have any subtrees,} \\ 1 & \text{if } \mathcal{F} \text{ is upward-closed,} \\ \max\{m^U_{\mathcal{F}_1}, m^U_{\mathcal{F}_2}, \ldots, m^U_{\mathcal{F}_k}\} & \text{if } \mathcal{C} \notin \mathcal{F} \text{ (Adam node),} \\ \sum_{i=1}^{k} m^U_{\mathcal{F}_i} & \text{if } \mathcal{C} \in \mathcal{F} \text{ (Eve node).} \end{cases}$$

Theorem 3.7 ([Cha07a]). *If Eve has an almost-sure strategy in a $2\frac{1}{2}$-player Muller game with the winning condition $\mathcal{F}$, she has a randomised almost-sure strategy with at most $m^U_{\mathcal{F}}$ memory states.* ∎

4. Randomised Upper Bound

The upper bound of Theorem 3.7 is not tight for all conditions. For example, the number $m^U_{\mathfrak{F}}$ of the condition $\mathfrak{F}$ in Figure 1(b) is three, while there is always an almost-sure strategy with two memory states. We present here yet another number for any Muller condition $\mathcal{F}$, denoted $r_{\mathcal{F}}$, that we compute from the Zielonka Tree:

Definition 4.1 (Number $r_{\mathcal{F}}$ of a Muller condition). Let $\mathcal{F} \subseteq \mathcal{P}(\mathcal{C})$ be a Muller condition, where the root has $k+l$ children, l of them being leaves. We denote by $\mathcal{Z}_{\mathcal{F}_1,\mathcal{C}_1}, \mathcal{Z}_{\mathcal{F}_2,\mathcal{C}_2}, \ldots, \mathcal{Z}_{\mathcal{F}_k,\mathcal{C}_k}$ the non-leaves subtrees attached to the root of $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$. We define $r_{\mathcal{F}}$ inductively as follows:

$$r_{\mathcal{F}} = \begin{cases} 1 & \text{if } \mathcal{Z}_{\mathcal{F},\mathcal{C}} \text{ does not have any subtrees,} \\ \max\{1, r_{\mathcal{F}_1}, r_{\mathcal{F}_2}, \ldots, r_{\mathcal{F}_k}\} & \text{if } \mathcal{C} \notin \mathcal{F} \text{ (Adam node),} \\ \sum_{i=1}^{k} r_{\mathcal{F}_i} & \text{if } \mathcal{C} \in \mathcal{F} \text{ (Eve node) and } l = 0, \\ \sum_{i=1}^{k} r_{\mathcal{F}_i} + 1 & \text{if } \mathcal{C} \in \mathcal{F} \text{ (Eve node) and } l > 0. \end{cases}$$

The first remark is that if $\emptyset \in \mathcal{F}$, $r_{\mathcal{F}}$ is equal to $m_{\mathcal{F}}$: as the leaves belong to Eve, the fourth case cannot occur. In the other case, the intuition is that we merge leaves if they are siblings. For example, the number $r_{\mathfrak{F}}$ for our recurring example is two: one for the leaves labelled *bcd* and *acd*, and one for the leaves labelled *a* and *b*. The number $m_{\mathfrak{F}}$ is four (one for each leaf), and $m^U_{\mathfrak{F}}$ is three (one for the leaves labelled *a* and *b*, and one for each other leaf). This section will be devoted to the proof of Theorem 4.2:

Theorem 4.2 (Randomised upper bound)**.** *If Eve has an almost-sure strategy in a 2-$\frac{1}{2}$ player Muller game with the winning condition $\mathcal{F}$), she has an almost-sure strategy with memory $r_{\mathcal{F}}$.*

Let $\mathcal{G} = (\mathcal{F}, \mathcal{A})$ be a game defined on the set of colours $\mathcal{C}$ such that Eve wins from any initial node. We describe in the next three subsections a recursive procedure to compute an almost-sure strategy for Eve with $r_{\mathcal{F}}$ memory states in each non-trivial case in the definition of $r_{\mathcal{F}}$. We use two lemmas — Lemmas 4.3 and 4.5 — that derive directly from similar results in [DJW97] and [Cha07b]. The application of these principles to the game $\mathfrak{G}$ in Figure 1 builds a randomised strategy with two memory states *left* and *right*. In *left*, Eve sends the token to ($\nwarrow$ *or* $\swarrow$) and in *right*, to ($\nearrow$ *or* $\searrow$). The memory switches from *right* to *left* with probability one when the token visits a *c*, and from *left* to *right* with probability $\frac{1}{2}$ at each step.

4.1. $\mathcal{C}$ is winning for Adam

In the case where Adam wins the set $\mathcal{C}$, the construction of σ relies on Lemma 4.3:

Lemma 4.3. *Let $\mathcal{F} \subseteq \mathcal{P}(\mathcal{C})$ be a Muller winning condition such that $\mathcal{C} \notin \mathcal{F}$, and $\mathcal{A}$ be a $2\frac{1}{2}$-player arena such that Eve wins everywhere. There are subarenas $\mathcal{A}_1 \dots \mathcal{A}_n$ such that:*

- $i \neq j \Rightarrow \mathcal{A}_i \cap \mathcal{A}_j = \emptyset$;
- $\forall i, \mathcal{A}_i$ *is a trap for Adam in the subarena* $\mathcal{A} \setminus \mathrm{Attr}_E\left(\cup_{j=1}^{i-1}\mathcal{A}_j\right)$;
- $\forall i, \chi(\mathcal{A}_i)$ *is included in the label E_i of a child of the root of $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$, and Eve wins everywhere in $(\mathcal{A}_i, \mathcal{F} \restriction E_i)$;*
- $\mathcal{A} = \mathrm{Attr}_E(\cup_{j=1}^{n}\mathcal{A}_j)$.

Let the subarenas $\mathcal{A}_i$ be the ones whose existence is proved in this lemma. We denote by σ_i the almost-sure strategy for Eve in $\mathcal{A}_i$, and by a_i the attractor strategy for Eve to $\mathcal{A}_i$ in the arena $\mathcal{A} \setminus \mathrm{Attr}(\cup_{j=1}^{i-1}\mathcal{A}_j)$. We identify the memory states of the σ_i, so their union has the same cardinal as the largest of them. For a state s, if $i = \min\{j \mid s \in \underset{E}{\mathrm{Attr}}(\cup_{k=1}^{j}\mathcal{A}_k)\}$, we define $\sigma(s, m)$ by:

- if $s \in \mathcal{A}_i$
  - $\sigma^{\mathtt{u}}(s, m) = \sigma_i^{\mathtt{u}}(s, m)$
  - $\sigma^{\mathtt{n}}(s, m) = \sigma_i^{\mathtt{n}}(s, m)$
- if $s \in \mathrm{Attr}_E(\cup_{k=1}^{i}\mathcal{A}_k) \setminus \mathcal{A}_i$
  - $\sigma^{\mathtt{u}}(s, m) = m$
  - $\sigma^{\mathtt{n}}(s, m) = a_i(s)$

By induction hypothesis over the number of colours, we can assume that the strategies σ_i have $r_{\mathcal{F}_i}$ memory states. The strategy σ uses $\max\{r_{\mathcal{F}_i}\}$ memory states.

Proposition 4.4. $\mathbb{P}^{\sigma,\tau}_{s_0}(\exists i, \mathrm{Inf}(\rho) \subseteq \mathcal{A}_i) = 1$.

Proof. The subarenas $\mathcal{A}_i$ are embedded traps, defined in such a way that the token can escape an $\mathcal{A}_i$ only by going to the attractor of a smaller one. Eve has thus a positive probability of reaching an $\mathcal{A}_j$ with $j < i$. Thus, if the token escapes one of the A_i infinitely often, the token has probability one to go to an $\mathcal{A}_j$ with $j < i$. By argument of minimality, after a finite prefix, the token will stay in one of the traps forever. ∎

The strategy σ_i is almost-sure from any state in $\mathcal{A}_i$. As Muller conditions are prefix-independent, it follows from Proposition 4.4 that σ is also almost-sure from any state in $\mathcal{A}$.

4.2. $\mathcal{C}$ is winning for Eve, and the root of $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$ has no leaves among its children.

In this case, the construction relies on the following lemma:

Lemma 4.5. *Let $\mathcal{F} \subseteq \mathcal{P}(\mathcal{C})$ be a Muller winning condition such that $\mathcal{C} \in \mathcal{F}$, $\mathcal{A}$ a $2\frac{1}{2}$-player arena coloured by $\mathcal{C}$ such that Eve wins everywhere, and A_i the label of a child of the root in $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$. Then, Eve wins everywhere on the subarena $\mathcal{A} \setminus \mathrm{Attr}_E(\chi^{-1}(\mathcal{C} \setminus A_i))$ with the condition $\mathcal{F} \restriction A_i$.*

Eve has a strategy σ_i that is almost-sure from each state in $\mathcal{A} \setminus \mathrm{Attr}_E(\chi^{-1}(\mathcal{C} \setminus A_i))$. In this case, the set of memory states of σ is $M = \cup_{i=1}^{k}(i \times M^i)$. The "next-move" and "memory-update" functions $\sigma^{\mathtt{n}}$ and $\sigma^{\mathtt{u}}$ for a memory state $m = (i, m^i)$ are defined below:

- if $s \in \chi^{-1}(\mathcal{C} \setminus A_i)$
  - $\sigma^{\mathtt{u}}(s, (i, m^i)) = (i + 1, m^{i+1})$ where m^{i+1} is any state in M^{i+1}
  - if $s \in S_E$, $\sigma^{\mathtt{n}}(s, (i, m^i))$ is any successor of s in $\mathcal{A}$
- if $s \in \mathrm{Attr}_E(\chi^{-1}(\mathcal{C} \setminus A_i))$
  - $\sigma^{\mathtt{u}}(s, (i, m^i)) = (i, m^i)$
  - $\sigma^{\mathtt{n}}(s, (i, m^i)) = a_i(s)$
- if $s \in \mathcal{A} \setminus \mathrm{Attr}_E(\chi^{-1}(\mathcal{C} \setminus A_i))$
  - $\sigma^{\mathtt{u}}(s, (i, m^i)) = (i, \sigma_i^{\mathtt{u}}(s, m^i))$
  - $\sigma^{\mathtt{n}}(s, (i, m^i)) = \sigma_i^{\mathtt{n}}(s, m^i)$

Once again, we can assume that the memory M_i of the strategy σ_i is of size $r_{\mathcal{F}\restriction A_i}$. Here, however, the memory set of σ is the disjoint union of the M_i's, so σ's needs the sum of the $\{r_{\mathcal{F}\restriction A_i}\}$'s.

Proposition 4.6. *Let $\mathfrak{uc}$ be the event "the top-level memory of σ is ultimately constant". Then, $\mathbb{P}_{s_0}^{\sigma,\tau}(\rho \in \Phi_{\mathcal{F}} \mid \mathfrak{uc}) = 1$.*

Proof. We call i the value of the top-level memory at the limit. After a finite prefix, the token stops visiting $\chi^{-1}(\mathcal{C} \setminus A_i)$. Thus, with probability one, it also stops visiting $\mathrm{Attr}_E(\chi^{-1}(\mathcal{C} \setminus A_i))$. From this point on, the token stays in the arena $\mathcal{A}_i$, where Eve plays with the almost-sure strategy σ_i. Thus, $\mathbb{P}^{\sigma,\tau}(\rho \in \Phi_{\mathcal{F}\restriction A_i} \mid \mathfrak{uc}) = 1$, and, as $\Phi_{\mathcal{F}\restriction A_i} \subseteq \Phi_{\mathcal{F}}$, Proposition 4.6 follows. ∎

Proposition 4.7. *If the top-level memory takes each value in $1 \dots k$ infinitely often, then surely, $\forall i \in 1 \dots k, \chi(\mathrm{Inf}(\rho)) \not\subseteq A_i$.*

Proof. The update on the top-level memory follows a cycle on $1 \dots k$, leaving i only when the token visits $\chi^{-1}(\mathcal{C} \setminus A_i)$. Thus, in order for the top-level memory to change continuously, the token has to visit each of the $\chi^{-1}(\mathcal{C} \setminus A_i)$ infinitely often. Proposition 4.7 follows. ∎

4.3. $\mathcal{C}$ is winning for Eve, and the root of $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$ has at least one leaf in its children.

As in the previous section, the construction relies on Lemma 4.5. In fact, the construction for children which are not leaves, labelled $A_1, \dots, A_k$, is exactly the same. The difference is that we add here a single memory state —0— that represents all the leaves (labelled $A_{-1}, \dots, A_{-l}$). The memory states are thus updated modulo $k+1$, and not modulo k. The "next-move" function of σ when the top-level memory is 0 is an even distribution over all the successors in A of the current state. The "memory-update" function has probability $\frac{1}{2}$ to stay into 0, and $\frac{1}{2}$ to go to $(1, m_1)$, for some memory state $m_1 \in M_1$. Thus, σ uses memory $\sum_{i=1}^{k} r_{\mathcal{F}_i} + 1$. We prove now that σ is almost-sure. The structure of the proof is the same as in the former section, with some extra considerations for the memory state 0.

Proposition 4.8. *Let* $\mathfrak{uc}$ *be the event "the top-level memory of* σ *is ultimately constant and different from 0". Then,* $\mathbb{P}_s^{\sigma,\tau}(\rho \in \Phi_{\mathcal{F}} \mid \mathfrak{uc}) = 1$.

Proof. The proof is exactly the same as the one of Proposition 4.6. ∎

Proposition 4.9. *The event "the top-level memory is ultimately constant and equal to 0" has probability 0.*

Proof. When the top-level memory is 0, the memory-update function has probability $\frac{1}{2}$ at each step to switch to 1. Proposition 4.9 follows. ∎

Proposition 4.10 considers the case where the top-level memory evolves continuously. By definition of the memory update, this can happen only if all the memory states are visited infinitely often.

Proposition 4.10. *Let* $\mathfrak{ec}$ *be the event "the top-level memory takes each value in* $0 \dots k$ *infinitely often". Then,* $\forall i \in -l \dots k, \mathbb{P}_s^{\sigma,\tau}(\chi(\mathrm{Inf}(\rho)) \subseteq \mathcal{C}_i \mid \mathfrak{ec}) = 0$.

Proof. As in the proof of Proposition 4.7, from the fact that the memory is equal to each of the $i \in 1 \dots k$ infinitely often, we can deduce that the token surely visits each of the $\mathcal{C} \setminus A_i$ infinitely often. We only need to show that, with probability one and for any $j \in 1 \dots l$, the set of limit states is not included in A_{-j}. The Zielonka Trees of the conditions $\mathcal{F} \restriction A_{-j}$ are leaves. This means that they are trivial conditions, where all the plays are winning for Adam. Consequently, in this case, Lemma 4.5 guarantees that $\mathrm{Attr}_E(\chi^{-1}(\mathcal{C} \setminus A_{-j}))$ is the whole arena. The definition of σ in the memory state (0) is to play legal moves at random. There is thus a positive probability that Eve will play according to the attractor strategy a_j long enough to guarantee a positive probability that the token visits $\chi^{-1}(\mathcal{C} \setminus A_{-j})$. To be precise, for any $s \in S$, this probability is greater than $(2 \cdot |S|)^{-|S|}$. Thus, with probability one, the token visits each $\chi^{-1}(\mathcal{C} \setminus A_{-j})$ infinitely often. Proposition 4.10 follows. ∎

The initial case, where the Zielonka tree is reduced to a leaf, is trivial: the winner does not depend on the play. Thus, Theorem 4.2 follows from Sections 4.1, 4.2, and 4.3.

5. Lower Bound

In this section, we consider lower bounds on memory, *i.e.* if we fix a Muller condition $\mathcal{F}$ on a set of colours $\mathcal{C}$, the minimal size of the memory set that is enough to define randomised almost-sure strategies for Eve on any arena coloured by the set $\mathcal{C}$. In his thesis, Majumdar showed the following theorem:

Theorem 5.1 ([Maj03]). *For any set of colours $\mathcal{C}$, there is a 2-player Muller game $\mathcal{G}_\mathcal{C} = (\mathcal{A}_\mathcal{C}, \mathcal{F}_\mathcal{C})$ such that Eve has an almost sure, but none of her almost-sure strategies have less than $\frac{|\mathcal{C}|}{2}!$ memory states.*

However, this is a general lower bound on *all* Muller conditions, while we aim to find specific lower bounds for *each* condition. We prove here that there is a lower bound for each Muller condition that matches the upper bound of Theorem 4.2:

Theorem 5.2. *Let $\mathcal{F}$ be a Muller condition on $\mathcal{C}$. There is a 2-player arena $\mathcal{A}_\mathcal{F}$ over $\mathcal{C}$ such that Eve has a sure strategy, but none of her almost-sure strategies have less than $r_\mathcal{F}$ memory states.*

As the construction of the upper bound was based on the Zielonka tree, the lower bound is based on the Zielonka *DAG*:

Definition 5.3. The Zielonka DAG $\mathcal{D}_{\mathcal{F},\mathcal{C}}$ of a winning condition $\mathcal{F} \subseteq \mathcal{P}(\mathcal{C})$ is derived from $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$ by merging the nodes which share the same label.

5.1. Cropped DAGs

The relation between $r_\mathcal{F}$ and the shape of $\mathcal{D}_{\mathcal{F},\mathcal{C}}$ is asymmetrical: it depends directly on the number of children of Eve's nodes, and not at all on the number of children of Adam's nodes. The notion of *cropped DAG* is the next logical step: a sub-DAG where Eve's nodes keep all their children, while each node of Adam keeps only one child:

Definition 5.4. A DAG $\mathcal{E}$ is a cropped DAG of a Zielonka DAG $\mathcal{D}_{\mathcal{F},\mathcal{C}}$ if and only if

- The nodes of $\mathcal{E}$ are nodes of $\mathcal{D}_{\mathcal{F},\mathcal{C}}$, with the same owner and label.
- There is only one node without predecessor in $\mathcal{E}$, which we call the root of $\mathcal{E}$. It is the root of $\mathcal{D}_{\mathcal{F},\mathcal{C}}$, if it belongs to Eve; otherwise, it is one of its children.
- The children of a node of Eve in $\mathcal{E}$ are exactly its children in $\mathcal{D}_{\mathcal{F},\mathcal{C}}$.
- A node of Adam has exactly one child in $\mathcal{E}$, chosen among his children in $\mathcal{D}_{\mathcal{F},\mathcal{C}}$, provided there is one. If it has no children in $\mathcal{D}_{\mathcal{F},\mathcal{C}}$, it has no children in $\mathcal{E}$.

Cropped DAG resemble Zielonka DAGs: the nodes belong to either Eve or Adam, and they are labelled by sets of states. We can thus compute the number $r_\mathcal{E}$ of a cropped DAG $\mathcal{E}$ in a natural way. In fact, this number has a more intuitive meaning in the case of cropped DAGs: if the leaves belong to Eve, it is the number of branches; if Adam owns the leaves, it is the number of branches with the leaf removed. Furthermore, there is a direct link between the cropped DAGs of a Zielonka DAG $\mathcal{D}_{\mathcal{F},\mathcal{C}}$ and the number $r_\mathcal{F}$:

Proposition 5.5. *Let $\mathcal{F}$ be a Muller condition on $\mathcal{C}$, and $\mathcal{D}_{\mathcal{F},\mathcal{C}}$ be its Zielonka DAG. Then there is a cropped DAG $\mathcal{E}^*$ such that $r_{\mathcal{E}^*} = r_\mathcal{F}$.* ∎

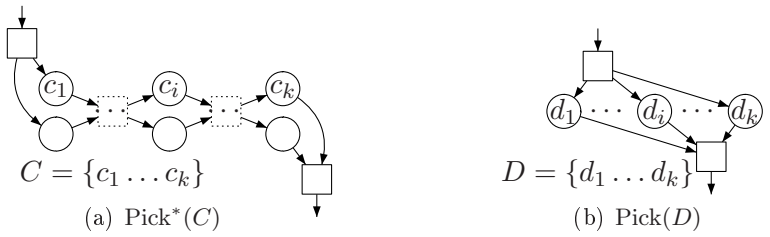
5.2. From cropped DAGs to arenas

From any cropped DAG $\mathcal{E}$ of $\mathcal{D}_{\mathcal{F},\mathcal{C}}$, we define an arena $\mathcal{A}_\mathcal{E}$ which follows roughly the structure of $\mathcal{E}$: the token starts from the root, goes towards the leaves, and then restarts from the root. In her nodes, Eve can choose to which child she wants to go. Adam's choices, on the other hand, consists in either stopping the current traversal or allowing it to proceed.

We first present two "macros", depending on a subset of $\mathcal{C}$:

- in Pick$^*(C)$, Adam can visit any subset of colours in C;
- in Pick(D), he must visit exactly one colour in D.

Both are represented in Figure 2, and they are the only occasions where colours are visited in $\mathcal{A}_\mathcal{E}$: all the other states are colourless.



(a) Pick$^*(C)$ (b) Pick(D)

Figure 2: Pick$^*(C)$ and Pick(D)

Eve's states in the arena $\mathcal{A}_\mathcal{E}$ are in bijection with her nodes in $\mathcal{E}$. Adam's nodes, on the other hand, are in bijection with the pairs parent-child of $\mathcal{E}$, where the parent belongs to Eve and the child to Adam.

In the state corresponding to the node n, Eve can send the token to any state of the form $n - c$. In states corresponding to leaves, Eve has no decision to take, and Adam can visit any colours in the label of the leaf (Pick* procedure). The token is then sent back to the root.

Adam's moves do not involve the choice of a child: by Definition 5.4, Adam's nodes in $\mathcal{E}$ have but one child. Instead, he can either stop the current traversal, or, if the current node is not a leaf, allow it to proceed to its only child. If he chooses to stop, Adam has to visit some coloured states before the token is sent back to the root. The available choices depend on the labels of both the current and the *former* nodes — which is why there are as many copies of Adam's nodes in $\mathcal{A}_\mathcal{E}$ as they have parents in $\mathcal{E}$. If the parent is labelled by E, and the current node by A, the token goes through Pick$^*(E)$ and Pick$(E \setminus A)$. Adam can thus choose any number of colours in E, as long as he chooses at least one outside of A.

(a) Edge "E" - "A" when "A" is a node (b) Edge "E" - "A" when "A" is a leaf

Figure 3: Adam's states in $\mathcal{A}_\mathcal{E}$.

5.3. Winning strategy and branch strategies

We first describe a sure strategy ς for Eve in the game $(\mathcal{A}_\mathcal{E}, \mathcal{F})$. Its memory states are the branches of $\mathcal{E}$, and do not change during a traversal. If the current memory state is

$b = E_1A_1 \dots E_\ell(A_\ell)$, Eve's moves follow the branch b: in E_i, she goes to $E_i - A_i$. When Adam stops the traversal at the ith step, Eve updates her memory as follows:

- If E_i has zero or one child in $\mathcal{E}$, the memory is unchanged;
- otherwise, the new memory branch has $E_1A_1 \dots E_iA$ as a prefix, where A is the next child of E_i, or the first one if A_i was the last.

Proposition 5.6. *The strategy ς is surely winning for Eve in the game $(\mathcal{A}_\mathcal{E}, \mathcal{F})$.*

Proof. Let ρ be a play consistent with ς. We denote by i the smallest integer such that traversals stops infinitely often at the ith step. After a finite prefix, the first $2i-1$ nodes in the memory branch are constant, and we denote them by $E_1A_1E_2 \dots E_i$. From this point on, the colours visited belong to E_i. Furthermore, each time a traversal stops at step i, a state is visited outside of the current A_i, which changes afterwards to the next, in a circular way. It follows that $\mathrm{Inf}(\rho) \subseteq E_i$, and, for any child A of E_i in $\mathcal{E}$, $\mathrm{Inf}(\rho) \not\subseteq A$. Thus ρ is winning for Eve. Proposition 5.6 follows. ∎

Obviously, Adam has no winning strategy in $\mathcal{A}_\mathcal{E}$. However, we describe the class of *branch strategies*, whose point is to punish any attempt of Eve to win with less than $r_\mathcal{F}$ memory states. There is one such strategy τ_b for each branch b in $\mathcal{E}$ (whence the name), and the principle is that τ_b stops the traversal as soon as Eve deviates from b:

Definition 5.7. The branch strategy τ_b for Adam in $\mathcal{A}_\mathcal{E}$, corresponding to the branch $b = E_1A_1E_2 \dots E_\ell(A_\ell)$ in $\mathcal{E}$, is a positional strategy whose moves are described below.

- In a state $E - A$ such that $\exists i, E = E_i \wedge A \neq A_i$: stop the traversal and visit A_i;
- in a state $E - A$ such that $\exists i, E = E_i \wedge A = A_i$: send the token to E_{i+1};
- in the state $E_\ell - A_\ell$, or the leaf E_ℓ: visit the colours of E_ℓ.

No move is given for a state $E - A$ such that $\forall i, E \neq E_i$, as these states are not reachable from the root when Adam plays τ_b. Notice also that when Adam chooses to stop a traversal in a state $E_i - A$, he *can* visit exactly the colours of A_i: as A and A_i are maximal subsets of E_i, there is at least one state in $A_i \setminus A$ that he can pick in the $\mathrm{Pick}(E_i \setminus A)$ area.

5.4. Winning against branch strategies

The key idea of the proof of Theorem 5.2 is that if two branches b and b' of $\mathcal{E}$ are too different, Eve needs different memory states to win against τ_b and $\tau_{b'}$.

Proposition 5.8. *Let $\sigma = (M, \sigma^\mathtt{n}, \sigma^\mathtt{u})$ be an almost-sure strategy for Eve in $(\mathcal{A}_\mathcal{E}, \mathcal{F})$. Then σ has memory at least $r_\mathcal{E}$.*

Proof. Let $b = E_1A_1 \dots E_\ell(A_\ell)$ be a branch of $\mathcal{E}$ and τ_b be the corresponding branch strategy for Adam. By definition of τ_b, the set of colours visited in a traversal consistent with τ_b is one of the A_i's, or E_ℓ if and only if Eve plays along b. As σ is almost-sure, there must be a memory state m such that Eve has a positive probability to play along b. It is also necessary to ensure that none of the A_i's is visited infinitely often, with the possible exception of A_ℓ. So, if Eve has a positive to play along a branch b' when she is in the memory state m, $E_1A_1 \dots E_\ell$ must be a prefix of b'. It follows that a single memory state can be suitable against two strategies τ_b and $\tau_{b'}$ with $b = E_1A_1 \dots E_\ell(A_\ell)$ and $b' = E'_1A'_1 \dots E'_{\ell'}(A'_{\ell'})$ only if $\ell = \ell'$ and $\forall i \leq \ell, E_i = E'_i$. By Definition 4.1, the underlying equivalence relation has $r_\mathcal{E}$ equivalence classes. Proposition 5.8 follows. ∎

By Proposition 5.5, there is a cropped DAG $\mathcal{E}$ of $\mathcal{D}_{\mathcal{F},\mathcal{C}}$ such that $r_{\mathcal{E}} = r_{\mathcal{F}}$. So, in general, Eve needs randomised strategies with memory $r_{\mathcal{F}}$ in order to win games whose winning condition is $\mathcal{F}$. This completes the proof of Theorem 5.2.

6. Conclusion

We have provided better and tight bounds for the memory needed to define almost sure winning randomised strategies. This allows us to characterise the class of Muller conditions which admit randomised memoryless strategies:

Corollary 6.1. *Eve admits randomised memoryless almost-sure strategies for a Muller condition $\mathcal{F}$ if and only if all her nodes in $\mathcal{Z}_{\mathcal{F},\mathcal{C}}$ have either one child, or only leave children.* ■

This yields a NP algorithm for the winner problem of such games, as solving $1\frac{1}{2}$-player Muller games is Ptime [CdAH04]. Another consequence of our result is that for each Muller condition, at least one of the players cannot improve its memory through randomisation:

Corollary 6.2. *Let $\mathcal{F}$ be a Muller condition. If $\emptyset \in \mathcal{F}$, Eve needs as much memory for randomised strategy as for pure strategies. Otherwise, Adam does.* ■

Our proof of lower bound also improves on the size of the witness arena: it is roughly equivalent to the size of the Zielonka DAG, instead of the size of the Zielonka tree. Whether these bounds still hold for arenas of size polynomial in the number of colours remains an open question, except for special cases like Streett games [Hor07].

References

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *TAMS*, 138:295–311, 1969.

[CdAH04] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Trading memory for randomness. In *QEST*, pages 206–217, 2004.

[Cha07a] Krishnendu Chatterjee. Optimal Strategy Synthesis in Stochastic Muller Games. In *FOSSACS*, pages 138–152, 2007.

[Cha07b] Krishnendu Chatterjee. Stochastic Müller Games are PSPACE-Complete. In *FSTTCS*, pages 436–448, 2007.

[deA97] Luca de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford, 1997.

[DJW97] Stefan Dziembowski, Marcin Jurdzinski, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110, 1997.

[GH82] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *STOC*, pages 60–65, 1982.

[Hor07] Florian Horn. Dicing on the Streett. *IPL*, 104(1):1–9, 2007.

[Maj03] Rupak Majumdar. *Symbolic Algorithms for Verification and Control.* PhD thesis, Berkeley, 2003

[MP92] Zohar Manna and Amir Pnueli. *The temporal Logic of Reactive and Concurrent Systems: Specification.* Springer, 1992.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *STACS*, pages 1–13, 1995.

[Zie98] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *TCS*, 200(1-2):135–183, 1998.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 553–564
www.stacs-conf.org

AMBIGUITY AND COMMUNICATION

JURAJ HROMKOVIČ [1] AND GEORG SCHNITGER [2]

[1] Department of Computer Science, ETH Zürich,
ETH Zentrum, CH-8022 Zürich, Switzerland
E-mail address: juraj.hromkovic@inf.ethz.ch

[2] Institut für Informatik, Goethe Universität,
Robert Mayer-Strasse 11-15, D-6054 Frankfurt a. M., Germany
E-mail address: georg@thi.informatik.uni-frankfurt.de

Abstract. The ambiguity of a nondeterministic finite automaton (NFA) N for input size n is the maximal number of accepting computations of N for an input of size n. For all $k, r \in \mathbb{N}$ we construct languages $L_{r,k}$ which can be recognized by NFA's with size $k{\cdot}\text{poly}(r)$ and ambiguity $O(n^k)$, but $L_{r,k}$ has only NFA's with exponential size, if ambiguity $o(n^k)$ is required. In particular, a hierarchy for polynomial ambiguity is obtained, solving a long standing open problem (Ravikumar and Ibarra, 1989, Leung, 1998).

1. Introduction

The ambiguity of an NFA N measures the degree of nondeterminism employed by N as a function of the input size: let $\text{ambig}_N(x)$ be the number of accepting computations of N on input x and define

$$\text{ambig}_N(n) = \max\{\text{ambig}_N(x) : x \in \Sigma^n\}$$

to be the ambiguity of N. There are related complexity measures such as the advice and the leaf complexity of N. To describe their definition let $T_N(x)$ be the computation tree of N on input x. Then $\text{advice}_N(x)$ is the maximum, over all paths in $T_N(x)$ from the root to a leaf, of the number of nodes with at least two children and

$$\text{advice}_N(n) = \max\{\text{advice}_N(x) : x \in \Sigma^n\}$$

is the advice complexity of N. The leaf complexity of N determines the maximal number of computations for inputs of length n. Thus, if $\text{leaf}_N(x)$ is the number of leaves of $T_N(x)$, then

$$\text{leaf}_N(n) = \max\{\text{leaf}_N(x) : x \in \Sigma^n\}.$$

For a minimal NFA N these measures are related as follows [2]

$$\text{advice}_N(n), \text{ambig}_N(n) \leq \text{leaf}_N(n) = O(\text{advice}_N(n) \cdot \text{ambig}_N(n))$$

Key words and phrases: Nondeterministic finite automata, ambiguity, communication complexity.

Supported by SNF-grant 200020-120073 and DFG-grant SCHN 503/4-1. Part of the work was done while the second author was visiting the ETH Zürich.

and, since $\text{advice}_N(n)$ is at most linear, leaf complexity and ambiguity are polynomially related, provided both are at least linear. Since leaf complexity is either bounded by a constant or at least linear but polynomially bounded, or otherwise exponential in the input length, we obtain that ambiguity is either bounded by a constant or bounded by a polynomial or at least exponential [2].

Advice and leaf complexity are rather coarse measures, since advice and leaf complexity of an unambiguous NFA may be linear. Ambiguity on the other hand also influences the tractability of algorithmic questions. For instance, for any fixed $k \in \mathbb{N}$ it can be determined efficiently whether two NFA's of ambiguity at most k are equivalent, resp. whether the ambiguity of a given NFA is at most k [7].

How large is the decrease in conciseness, i.e., the increase in the number of states, if ambiguity is restricted? To study this question, four classes of NFA's, namely UNA (unambiguous nondeterministic automata), FNA (finitely ambiguous NFA), PNA (polynomially ambiguous NFA) and ENA (exponentially ambiguous NFA) are introduced in [6]. The classification into FNA's, PNA's or ENA's can be performed efficiently [8].

Remember that the ambiguity of an NFA N is either at least exponential or at most polynomial and hence an NFA is either a PNA or an ENA. Leung [4] shows that there are ENA's N_n with n states such that any equivalent PNA has at least $2^n - 1$ states. Hence ENA's can be far more succinct than PNA's. Subsequently a similar result, applicable to a larger class of languages, was shown in [2] by using methods of communication complexity. In particular, the conciseness problem for PNA's can be reduced to the following communication result for the iterated language of non-disjointness. Let Σ_r be the alphabet of all subsets of $\{1, \ldots, r^{32}\}$ of size r and set

$$L_r = \{xy | x, y \in \Sigma_r \text{ and } x \cap y \neq \emptyset\}.$$

Thus $(L_r)^t$ consists of all strings $x_1 y_1 \cdots x_t y_t$ where all pairs $x_i y_i$ correspond to overlapping subsets. We assume the standard communication model with two players, Alice and Bob, where Alice receives $x_1 \cdots x_t$ and Bob receives $y_1 \cdots y_t$. (Observe that $(L_r)^t$ has small NFA's with $\text{poly}(r + t)$ states.)

Fact 1.1. *([3], pages 51-53). Let $r, t \in \mathbb{N}$ be arbitrary. If a deterministic protocol D accepts only strings from $(L_r)^t$ and if at most $2^{\alpha \cdot r \cdot t}$ messages are exchanged, then D accepts at most $|(L_r)^t|/2^{\alpha \cdot t}$ strings from $(L_r)^t$. (α is a sufficiently small constant).*

Of particular interest are FNA's, for instance since their equivalence problem is efficiently solvable. However a separation of FNA's and PNA's has remained open for almost twenty years [4, 6]. We are able to show such a separation and even prove a hierarchy result for polynomial ambiguity. To describe our result we introduce the languages used in the separation. For a language L of strings of identical length define

$$\exists_k(L) \quad = \quad \{w_1 w_2 \cdots w_m \mid m \in \mathbb{N} \text{ and } w_i \in L \text{ for at least } k \text{ different positions}\}.$$

Thus the input is partitioned into blocks of identical length and an input is accepted iff at least k blocks belong to the finite set L. Now assume that L can be recognized by a small NFA N. Since L is a finite set, we can recognize $\exists_k(L)$ by an NFA with ambiguity $O(n^k)$, if we increase the size of N by at most the factor k.

How should the languages L look like? In a first attempt set $L = \{uv \mid u, v \in \{0,1\}^r, u \neq v\}$ as the language of inequality between r-bit strings. Then L is recognizable by an NFA with $\text{poly}(r)$ states and (bounded) ambiguity r. But $\exists_1(L)$ is also recognizable with $\text{poly}(r)$

states and ambiguity r: guess a position $i \in \{1, \ldots, r\}$ and accept $u^1 v^1 \cdots u^m v^m$ if $u_i^j \neq v_i^j$ for some $1 \leq j \leq r$.

What went wrong? Few advice bits suffice and these advice bits can be remembered. In our second (and successful) attempt we therefore set $L = (L_r)^t$, where we work with the iterated language of non-disjointness from Fact 1.1. This construction has two advantages. Firstly, L has a small NFA. Secondly, at least intuitively, the number of guesses required for L increases exponentially with t and hence a small NFA's for $\exists_1(L_r)$ cannot remember sequences of t guesses. Our main result verifies this intuition.

Theorem 1.2. *Let $r \in \mathbb{N}$ be arbitrary. Set $t = r^{1/3}$ and $L = (L_r)^t$. Any NFA for $\exists_k(L)$ with ambiguity $o(n^k)$ has at least $2^{\Omega((r/k^2)^{1/3})}$ states. However, $\exists_k(L)$ can be recognized by an NFA with ambiguity $O(n^k)$ and size $k \cdot \mathrm{poly}(r)$.*

Observe that we have obtained the claimed separation of FNA's and PNA's for $k = 1$, but Theorem 1.2 also establishes a hierarchy of polynomial ambiguity.

2. A Proof Sketch

We start by proving Theorem 1.2 for $k = 1$. Let $L = (L_r)^t$ and assume that the NFA N recognizes $\exists_1(L)$ with sublinear ambiguity. Observe that all strings in L have length $2t$ and hence strings in $\exists_1(L)$ have blocks of identical length $2t$. We set $K = \Sigma_r^{2t}$, where Σ_r is the alphabet of L_r. Finally set

$$\exists_{=0}(L) \quad = \quad \{w_1 w_2 \cdots w_m : m \in \mathbb{N} \text{ and } w_i \in K \setminus L \text{ for all } i \}.$$

Thus, as in the definition of $\exists_1(L)$, the input is partitioned into blocks and an input is accepted iff *no* block belongs to the finite set L. The computationally hardest task for the NFA N is to separate $\exists_{=0}(L)$ from $\exists_1(L)$.

The critical part of the argument is to exploit the limitation of sublinear ambiguity. Let Q be the set of states of N. In Section 3 we construct states $p_0, p_1 \in Q$ such that at least $|L|/|Q|^2$ strings in L have a computation starting in p_0 and ending in p_1. Moreover we show in Lemma 3.3 that for any string $z' \in K \setminus L$ there is a string $u \in \exists_{=0}(L)$ such that strings $S(z')$ with period $z'u$ can be "stored" in a "launching cycle" before reaching p_0 and in a "storage cycle" after leaving p_1. The launching cycle has the form $r \xrightarrow{(z'u)^a} r$ and allows to reach p_0 via a computation $r \xrightarrow{(z'u)^{a_1}} p_0$; analogously the storage cycle is built from computations $p_1 \xrightarrow{(uz')^{a_2}} s$ and $s \xrightarrow{(uz')^a} s$. So far the launching cycle is harmless, since it delivers strings in $\exists_{=0}(L)$ to state p_0, but these strings cannot use computations from p_0 to p_1 which may be reserved for strings in L. However, if a single occurrence of z' within $S(z')$ is replaced by an impostor string $z \in L$ and if the launching cycle does not detect the replacement, then N is forced into linear ambiguity, provided the impostor z can also hide at a matching position within the storage cycle (see Lemma 3.4).

Thus the NFA N has to solve the "detection problem", namely it has to detect whether an impostor $z \in L$ has replaced an occurrence of $z' \in K \setminus L$ in both cycles. The detection problem is set up in such a way that

- at least $|L|/|Q|^2$ strings from L are accepted, namely those strings $z \in L$ with a computation $p_0 \xrightarrow{z} p_1$, and

- all strings z which for some $z' \in K \setminus L$ survive in matching positions within both cycles are rejected. In particular, all strings in $K \setminus L$ are rejected, since a string $z \in K \setminus L$ is its own impostor.

Observe that no string z is simultaneously accepted as well as rejected, since all impostors have to be detected. N may try to solve the detection problem unconventionally for instance by allowing a potential impostor z to survive undetected within the launching and storage cycle, but not allowing z to survive in *matching positions* within both cycles. Also N does not have to solve the detection problem completely, since it can tolerate an impostor z without a computation $p_0 \xrightarrow{z} p_1$.

We then simulate N in Section 4 by a nondeterministic communication protocol which rejects all strings in $K \setminus L$, accepts at least $|L|/|Q|^2$ strings in L and does not simultaneously accept and reject a string in $K \setminus L$ (see Lemma 4.1). Thus we have reduced the problem of avoiding linear ambiguity for NFA's recognizing $\exists_1(L)$ to a communication problem in which a rather small minority of strings in L has to be separated from all of $K \setminus L$. We show in Lemma 5.1 how to transform such a nondeterministic protocol into a deterministic protocol by increasing the number of messages only subexponentially. We are left with a deterministic protocol which rejects all strings in $K \setminus L$ and accepts at least $|L|/|Q|^2$ strings in L. Finally the argument concludes with an application of Fact 1.1. Thus, as in the case of exponential ambiguity, we again have reduced the conciseness problem to an investigation of deterministic protocols which recognize a "small, but significant chunk" of a given product language.

The general case of ambiguity $O(n^k)$ is tackled in Section 6. Showing the existence of launching and storage cycles has now become a more complex problem. Previously it was sufficient that the periodic string $S(z)$ was "living" in the one launching and the one storage cycle. Now we have to work with a vector $p_0, p_1, \ldots, p_{2k-2}, p_{2k-1}$ of states and have to move $S(z)$ to p_0 and all the way from p_{2i+1} to $p_{2(i+1)}$ for all $i = 0, \ldots k-2$ and finally from p_{2k-1} to an accepting state.

3. From Automata to Communication

We begin by utilizing the special structure of the languages $\exists_1(L)$.

Definition 3.1. Let N be an NFA for $\exists_1(L)$ with initial state q_0. Let p be an arbitrary state of N.

(a) We say that a string $v \in \exists_{=0}(L)$ **reaches** state p iff there is a string $u \in \exists_{=0}(L)$ and a computation for $u \cdot v$ which starts in q_0 and ends in p. Moreover state p **accepts** $v \in \exists_{=0}(L)$ iff there is a string $w \in \exists_{=0}(L)$ and an accepting computation for $v \cdot w$ starting in p.

(b) A pair (p_0, p_1) of states of N is **critical** for the pair $(\xi_0, \xi_1) \in \exists_{=0}(L) \times \exists_{=0}(L)$ iff all strings in $\exists_{=0}(L) \cdot \xi_0$ reach p_0 and all strings in $\xi_1 \cdot \exists_{=0}(L)$ are accepted by p_1.

Our next goal is to construct a pair $(\xi_0, \xi_1) \in \exists_{=0}(L) \times \exists_{=0}(L)$ such that for all strings $u\xi_0 z \xi_1 w \in \exists_{=0}(L) \cdot (\xi_0 \cdot L \cdot \xi_1) \cdot \exists_{=0}(L)$ acceptance is "decided" by critical pairs. In particular we construct (ξ_0, ξ_1) such that there are accepting computations of the form $q_0 \xrightarrow{u\xi_0} p_0 \xrightarrow{z} p_1 \xrightarrow{\xi_1 w} q_f$ for a final state q_f and a critical pair (p_0, p_1) for (ξ_0, ξ_1). The crucial advantage of a critical pair is that all strings in $\exists_0(L) \cdot \xi_0$ reach p_0 and all strings in $\xi_1 \cdot \exists_0(L)$ are accepted

by p_1; in particular, there is no transition $p_0 \xrightarrow{z} p_1$ for a string $z \in \exists_0(L)$ and acceptance is indeed decided by (p_0, p_1).

Lemma 3.2. *Let N be an NFA for $\exists_1(L)$. Then there are strings $\xi_0, \xi_1 \in \exists_{=0}(L)$ such that*

$$\bigcup_{(p_0,p_1) \text{ is critical for } (\xi_0,\xi_1)} \{z \in L \mid p_0 \xrightarrow{z} p_1\} = L.$$

Proof. We process the states of N in two phases. In the first phase we construct a string $\xi_0 \in \exists_{=0}(L)$ such that each state p is either *alive* for ξ_0 (i.e., all strings in $\exists_{=0}(L) \cdot \xi_0$ reach p) or *dead* for ξ_0 (i.e., no string in $\exists_{=0}(L) \cdot \xi_0$ reaches p). The construction process proceeds iteratively by processing all states p of N in an arbitrary order. We begin by setting $\xi_0 = \epsilon$. When processing state p we differentiate two cases.

Case 1: All strings in $\exists_{=0}(L) \cdot \xi_0$ reach p. We do not modify ξ_0. Observe that p is alive for ξ_0 and stays alive for any string in $\exists_{=0}(L)$ with suffix ξ_0.

Case 2: There is a string $\xi \in \exists_{=0}(L)$ such that $\xi \cdot \xi_0$ does not reach p. The string $\xi \cdot \xi_0$ does not reach p and hence no string in $\exists_{=0}(L) \cdot \xi \cdot \xi_0$ has a computation beginning in the starting state q_0 and ending in p. We replace ξ_0 by $\xi \cdot \xi_0$ and p is dead for ξ_0, but also dead for any string in $\exists_{=0}(L)$ with suffix ξ_0. Also observe that any already processed state q stays alive, resp. remains dead.

In the second phase we proceed completely analogously, but now construct a string $\xi_1 \in \exists_{=0}(L)$ such that each state p is either *alive* for ξ_1 (i.e., p accepts all strings in $\xi_1 \cdot \exists_{=0}(L)$) or *dead* for ξ_1 (i.e., p does not accept any string in $\xi_1 \cdot \exists_{=0}(L)$).

Now consider any string $s = \xi_0 z \xi_1$ in $M = \xi_0 \cdot L \cdot \xi_1$. Observe that M is a subset of $\exists_1(L)$. However ξ_0 cannot reach a dead state for ξ_0 and ξ_1 cannot be accepted by a dead state for ξ_1. Thus any accepting computation for s has to utilize a transition $p_0 \xrightarrow{z} p_1$ between alive states p_0 for ξ_0 and p_1 for ξ_1. But any pair (p_0, p_1) of alive states is a critical pair and we are done. ∎

From now on we fix a pair $(\xi_0, \xi_1) \in \exists_{=0}(L) \times \exists_{=0}(L)$ for which Lemma 3.2 holds. Let (p_0, p_1) be an arbitrary critical pair for (ξ_0, ξ_1). We now utilize that all strings in $\exists_{=0}(L) \cdot \xi_0$ reach p_0 and all strings in $\xi_1 \cdot \exists_{=0}(L)$ are accepted by p_1.

Lemma 3.3. *For all strings $z \in K \setminus L$ there are states r, s, integers $a \geq 1, a_1, a_2$ (with $a_1 + a_2 \leq a$) and a string $u \in \exists_{=0}(L)$ as well as computations*

$$r \xrightarrow{(zu)^a} r \xrightarrow{(zu)^{a_1}} p_0 \quad \text{and} \tag{3.1}$$

$$p_1 \xrightarrow{(uz)^{a_2}} s \xrightarrow{(uz)^a} s. \tag{3.2}$$

Proof. We consider all strings of the form

$$\alpha(z) = (z\xi_1\xi_0)^{|Q|} \quad \text{and} \quad \beta(z) = (\xi_1\xi_0 z)^{|Q|}.$$

The string $\alpha(z)$ has suffix ξ_0 and hence $\alpha(z)$ reaches p_0. As a consequence there is $\xi \in \exists_{=0}(L)$ and a computation C for $\xi \cdot \alpha(z)$ which begins in the initial state q_0 and reaches p_0. After reading ξ, computation C processes $\alpha(z)$ and produces a sequence of $|Q| + 1$ states, where we list all states before reading a copy of $z\xi_1\xi_0$, resp. after reading the last copy. A state r of N appears twice in this sequence and we obtain a transition of the form $r \xrightarrow{(z\xi_1\xi_0)^a} r$ for $a \geq 1$. Finally C, starting in r, reaches p_0 after reading the remaining a_1 copies.

To establish (3.1), we set $u = \xi_1\xi_0$ and obtain transitions $r \xrightarrow{(zu)^a} r$ and $r \xrightarrow{(zu)^{a_1}} p_0$. Thus (3.1) follows. Part (3.2) is established by a similar argument, but now applied to $\beta(z)$. This

time we get transitions $p_1 \stackrel{(uz)^{a_2}}{\to} s$ and $s \stackrel{(uz)^b}{\to} s$. But then $r \stackrel{(zu)^{ma}}{\to} r$ as well as $s \stackrel{(uz)^{m'b}}{\to} s$ are transitions for any multiples $m, m' \geq 1$ and the claim follows, if we replace both a and b by $ab(a_1 + a_2) \geq a_1 + a_2$. ∎

Let (p_0, p_1) be a critical pair for (ξ_0, ξ_1). We now introduce the detection problem for (p_0, p_1) in which strings in L have to be "weakly" separates from strings in $K \setminus L$. It turns out that any NFA N for $\exists_1(L)$ solves the detection problems for all critical pairs, provided N has ambiguity $o(n)$. Since we show later that N can be efficiently simulated by a communication protocol –with communication resources related to the number of states– and that the detection problem is hard for communication complexity, N must have many states. The detection problem of (p_0, p_1) has the following form:

(a) Accept a string $z \in K$ iff there is a computation $p_0 \stackrel{z}{\to} p_1$ of N. Remember that for no $z \in K \setminus L$ there is a computation

$$q_0 \stackrel{\xi\xi_0}{\to} p_0 \stackrel{z}{\to} p_1 \stackrel{\xi_1\xi'}{\to} q_f$$

with the initial state q_0, a final state q_f and strings $\xi, \xi', \xi_0, \xi_1 \in \exists_{=0}(L)$. Hence no string $z \in K \setminus L$ is accepted.

(b) Reject a string $z \in K$ iff there are states r, r', r'', s, s', s'', integers $a \geq 1, a_1, a_2$ (with $a_1 + a_2 \leq a$) and strings $u \in \exists_{=0}(L), z' \in K \setminus L$ with computations

$$r \stackrel{(z'u)^{a_1}}{\to} r' \stackrel{zu}{\to} r'' \stackrel{(z'u)^{a-a_1-1}}{\to} r \stackrel{(z'u)^{a_1}}{\to} p_0 \text{ and} \tag{3.3}$$

$$p_1 \stackrel{(uz')^{a_2}}{\to} s \stackrel{(uz')^{a-a_2-1}}{\to} s' \stackrel{uz}{\to} s'' \stackrel{(uz')^{a_2}}{\to} s. \tag{3.4}$$

(The computations (3.3) and (3.4) will be used later to define a launching and storage cycle respectively. It turns out that z is placed within matching positions of the $z'u$- and uz'-cycle and hence z plays the role of an impostor of z'.)

(c) $z \in K$ is left undecided iff z is neither accepted nor rejected.

To explain the purpose of these transitions consider the string

$$S_1 = [zu \cdot (z'u)^{a-1}] \cdot [zu \cdot (z'u)^{a-1}].$$

If we process the first half $zu \cdot (z'u)^{a-1}$ of S_1 starting in state r', then there is a computation C_0 of the form

$$r' \stackrel{zu}{\to} r'' \stackrel{(z'u)^{a-a_1-1}}{\to} r \stackrel{(z'u)^{a_1}}{\to} r'$$

as well as a computation C_1 from r' to p_0 according to (3.3). When reading the second half of S_1, computation C_0 splits into a computation C_{00} which goes full circle reaching state r' again and a computation C_{01} which reaches p_0 after completely reading S_1. Now assume that there is a transition $p_0 \stackrel{z}{\to} p_1$. Computation C_1 has reached p_0 after reading the first half of S_1 and now reads the second half $zu \cdot (z'u)^{a-1} = z \cdot (uz')^{a-1} \cdot u$ of S_1. It travels from p_0 to p_1 and subsequently reaches state s'', if *additionally* the string z is read. We have been successful

(1) in "storing" a mother computation via computation C_{00} in state r',
(2) preparing for a new "launch" in state p_0 via computation C_{01} and
(3) "storing" offspring computations in state s'' via computation C_1.

We utilize properties (1)-(3) by defining a sequence $(S_m \mid m \geq 1)$ with many computations, namely we set

$$S_{m+1} = S_m \cdot [zu \cdot (z'u)^{a-1}] = S_m \cdot [z \cdot (uz')^{a-1} \cdot u].$$

Assume inductively that there are computations for S_m which have reached the states r' and p_0 respectively and a computation for $S_m \cdot z$ which has reached s''. After reading the suffix $zu \cdot (z'u)^{a-1}$ of S_{m+1}, the computation starting in r' has split into a computation reaching r' again and a computation reaching p_0, whereas the freshly launched computation reaches s'' from p_0 after reading $S_{m+1} \cdot z$. Observe that all previously launched computations go full circle after reading $(uz')^{a-1} \cdot uz$ and again have reached state s''. As a consequence, there are m distinct computations for $S_m z$ all reaching state s'' at the same time.

We say that N has no redundant states, if each state is part of some accepting computation of N. Which strings are rejected and which strings are accepted?

Lemma 3.4. *Let N be an NFA recognizing $\exists_1(L)$ without redundant states. Also assume that N has ambiguity $o(n)$.*

(a) *Consider the detection problem of an arbitrary critical pair (p_0, p_1). Then all strings in $K \setminus L$ are rejected and no string in K is simultaneously accepted and rejected.*
(b) *Each string in L is accepted in the detection problem of some critical pair.*

Proof. (a) We observe first that every string $z \in K \setminus L$ is rejected. Why? We may choose $z' = z$ and the transitions required in (3.3) and (3.4) exist as a consequence of Lemma 3.3: the states r', r'' and s', s'' belong to the r-cycle and the s-cycle respectively.

Now assume that there is a string $z \in K$ which is accepted and rejected. Since z is accepted, there is a computation $p_0 \xrightarrow{z} p_1$. Also, since z is rejected, there are computations of the form (3.3) and (3.4). Thus we may construct the strings $S_m z$ for every m and obtain m distinct computations which, starting from state r', reach state s'' at the same time. But N does not have redundant states and each state, and in particular state r', is reachable from the initial state. Also each state, and in particular state s'', can reach an accepting state. Thus there are strings ξ_0, ξ_1 such that $\xi_0 \cdot S_m z \cdot \xi_1$ has m accepting computations. But $S_m z$ is a string with length linear in m and hence N has at least linear ambiguity.

(b) follows from part (a), if we apply Lemma 3.2. ∎

4. The Communication Problem

We show that the detection problem has an efficient communication protocol, provided a small NFA N with ambiguity $o(n)$ recognizes $\exists_1(L)$. Remember that $L = (L_r)^t$ and $K = \Sigma_r^{2t}$. We work with the conventional two-party communication model consisting of two players Alice and Bob. If $x_1 y_1 \cdots x_t y_t$ is the input of N, then Alice receives $x_1 \cdots x_t$ and Bob receives $y_1 \cdots y_t$ as their respective inputs. Alice and Bob communicate nondeterministically with computations either being accepting, rejecting or undecided. We say that an input is accepted if at least one computation is accepting, rejecting if at least one computation is rejecting and undecided if all computations are undecided. (Thus undecided computations play the role of rejecting computations for conventional nondeterminism.) Observe that we allow to simultaneously accept and reject an input.

Now assume that the NFA N recognizes $\exists_1(L)$. Let q, q^* be two states of N and let $z \in K$ be an input string. Our first goal is to determine whether N has a computation for z starting in q and ending in q^*. Set $q_0 = q$. Beginning with $i = 1$, Alice simulates N for input x_i by starting in state q_{i-1} and sends state q_i', if q_i' is reached. Bob simulates N for input y_i by starting in state q_i' and sends state q_i, if q_i is reached. In the last round Bob accepts if additionally $q_s = q^*$ holds and otherwise outputs "undecided". Obviously the

simulating protocol exchanges at most $|Q|^{2t}$ messages. It has an accepting computation iff N has a computation $q \xrightarrow{z} q^*$ and otherwise leaves the input undecided.

We say that a protocol solves the detection problem of (p_0, p_1) if the protocol labels each input as accepted, rejected or undecided as prescribed by the detection problem.

Lemma 4.1. *Assume that N recognizes $\exists_1(L)$ and that N has ambiguity $o(n)$. Let (p_0, p_1) be a critical pair for (ξ_0, ξ_1). Then there is a nondeterministic protocol P which solves the detection problem of (p_0, p_1) with $|Q|^{O(t)}$ messages.*

Proof. We begin by describing the protocol P. In its first attempt P tries to accept its input $z \in K$ by simulating the automaton N when reading z starting in state p_0. P accepts z iff state p_1 is reached and otherwise leaves z undecided.

In its second attempt P tries to reject z. Alice guesses states r, r', r'', s, s', s'' as well as strings $z' \in K \setminus L, u \in \exists_{=0}(L)$ and integers a_1, a_2, a (with $a_1 + a_2 \leq a$). Then Alice verifies the following transitions *without* communication, namely

- $r \xrightarrow{(z'u)^{a_1}} r'$ as well as $r'' \xrightarrow{(z'u)^{a-a_1-1}} r \xrightarrow{(z'u)^{a_1}} p_0$ and
- $p_1 \xrightarrow{(uz')^{a_2}} s \xrightarrow{(uz')^{a-a_2-1}} s'$ as well as $s'' \xrightarrow{(uz')^{a_2}} s$.

In order to check the remaining transition $r' \xrightarrow{zu} r''$ and $s' \xrightarrow{uz} s''$, Alice guesses additional states ρ, σ and verifies the transitions $\rho \xrightarrow{u} r''$ and $s' \xrightarrow{u} \sigma$ by herself. Subsequently Alice communicates the states r', ρ as well as σ, s'' and both Alice and Bob simulate the automaton N on input z for starting states r' and σ. Bob rejects iff the transitions $r' \xrightarrow{z} \rho$ and $\sigma \xrightarrow{z} s''$ have been verified and otherwise labels z as undecided. Observe that P exchanges at most $|Q|^{O(t)}$ messages, since P uses messages only when simulating N on the string $z \in K$. ∎

5. From Nondeterminism to Determinism

In Lemma 4.1 we have solved the detection problem of a critical pair by a nondeterministic protocol P with only $|Q|^{O(t)}$ messages. However the detection problem separates L from its complement $K \setminus L$ only weakly, since the majority of strings from L are either rejected or left undecided. We begin our analysis by transforming the nondeterministic protocol P into a deterministic protocol D. We avoid an exponential blowup in the number of messages by observing the structural limitations of P. In particular, P accepts a subset L_{yes} of L and rejects a superset L_{no} of $K \setminus L$, where L_{yes} and L_{no} are disjoint.

Lemma 5.1. *There is a deterministic protocol D which accepts at least $|L|/|Q|^2$ strings from L and rejects all strings from $K \setminus L$. No string is left undecided and no string is accepted as well as rejected. Moreover, at most $|Q|^{O(t^2 \cdot \log_2 |Q|)}$ messages are exchanged.*

Proof. We begin by fixing a critical pair (p_0, p_1) such that at least $|L|/|Q|^2$ strings are accepted in the detection problem of (p_0, p_1). Observe that such a critical pair exists as a consequence of Lemma 3.4 (b), since each string in L is accepted in the detection problem of at least one critical pair and there are at most $|Q|^2$ critical pairs.

Let L_{yes} be the subset of L which is accepted in the detection problem of (p_0, p_1) and let L_{no} be the superset of $K \setminus L$ of rejected strings. According to Lemma 4.1 there is a nondeterministic protocol P which solves the detection problem of (p_0, p_1) with at most $|Q|^{O(t)}$ messages. Thus there are conventional nondeterministic protocols P_{yes} for L_{yes} and P_{no} for L_{no} which exchange at most $|Q|^{O(t)}$ messages each.

To obtain a deterministic protocol D from P_{yes} and P_{no} we utilize that deterministic protocols with $M^{O(\log_2 M)}$ messages can be built from nondeterministic protocols, provided the protocols recognize a language *and* its complement by exchanging at most M messages [1]. Our situation however is more complicated, since L_{yes} is only a subset of the complement of L_{no}. We employ the construction in [5] with the following modifications. Define the communication matrix C of $(P_{\text{yes}}, P_{\text{no}})$ by setting

$$C[x_1 \cdots x_t, y_1 \cdots y_t] = \begin{cases} 1 & x_1 y_1 \cdots x_t y_t \in L_{\text{yes}}, \\ 0 & x_1 y_1 \cdots x_t y_t \in L_{\text{no}} \\ \text{undecided} & \text{otherwise.} \end{cases}$$

Each message m corresponds to a submatrix M of C defined by the collection of rows for which the message is sent and the collection of columns for which it is accepted. Now let M be a submatrix of the communication matrix C. We define $\Delta_{\text{yes}}(M)$ to be the maximal size of a submatrix T of M, where T, after a suitable permutation of rows and columns of M, is a lower triangular matrix with ones on the diagonal and zeroes above the diagonal. (Observe that T may contain undecided entries, but these entries have to appear below the diagonal.) Since L_{yes} is accepted by the nondeterministic protocol P_{yes} and since no two diagonal entries can be accepted by the same message, we obtain that $\Delta_{\text{yes}}(C)$ is bounded by the number of messages of P_{yes} and hence $\Delta_{\text{yes}}(C) \leq |Q|^{O(t)}$ follows.

We first try to reject the given input by deterministically selecting a sequence m_i of messages from the protocol P_{no}. As for the conventional transformation to deterministic protocols, the triangular message complexity will be halved in each step and in particular $\Delta_{\text{yes}}(M_1 \cap \cdots \cap M_i) \leq \Delta_{\text{yes}}(C)/2^i$ follows. We proceed as in the conventional transformation and stop the communication prematurely, if the output "no" can be excluded and output "yes". Otherwise, after at most $\log_2 \Delta_{\text{yes}}(C)$ rounds, we obtain $\Delta_{\text{yes}}(M_1 \cap \cdots \cap M_i) \leq 1$. As a consequence, the submatrix $M_1 \cap \cdots \cap M_i$ has no triangular submatrix of size two or larger. In particular, the submatrix M of $M_1 \cdots M_i$ spanned by all rows and columns of M_i^* with a one, contains all ones of $M_1 \cdots M_i$, no zeroes and possibly undecided entries. If the joint input belongs to M, then we stop and accept, resp. stop and reject otherwise. In each round only messages of P_{no} and hence at most $|Q|^{O(t)}$ messages are exchanged. Thus overall at most $\left[|Q|^{O(t)}\right]^{\log_2 \Delta_{\text{yes}}(C)} = |Q|^{O(t^2 \cdot \log_2 |Q|)}$ messages are generated. ∎

Remember that $L = (L_r)^t$, where L_r is the language of non-disjointness for r-element subsets of $\{1, \ldots, r^{32}\}$. Let D be a deterministic protocol which accepts only strings in L. Also let α be a sufficiently small positive constant. We apply Fact 1.1 and obtain that D accepts at most $|L|/2^{\alpha \cdot t}$ strings from L, provided at most $2^{\alpha \cdot r \cdot t}$ messages are exchanged.

Now, if an NFA N with sublinear ambiguity recognizes $\exists_1(L)$, then we apply Lemma 5.1 to obtain a deterministic protocol which exchanges at most $|Q|^{O(t^2 \cdot \log_2 |Q|)}$ messages, accepts at least $|L_r|/|Q|^2$ strings and accepts only strings from L. Thus, if $|Q|^{O(t^2 \cdot \log_2 |Q|)} \leq 2^{\alpha \cdot r \cdot t}$ for a sufficiently small positive constant α, then at most $|L|/2^{\alpha \cdot t}$ inputs from L are accepted. But the nondeterministic protocol accepts at least $|L|/|Q|^2$ strings from L and hence

$$|Q| = 2^{\Omega(t)} \tag{5.1}$$

follows. We set $t = r^{1/3}$. Let β be a sufficiently small positive constant. Now either $|Q| \geq 2^{\beta \cdot \sqrt{r/t}}$ and we are done, since then $|Q| = 2^{\Omega(r^{1/3})}$ or $|Q| < 2^{\beta \cdot \sqrt{r/t}}$ holds. In the latter case

$$|Q|^{t^2 \cdot \log_2 |Q|} < 2^{(\beta \cdot \sqrt{r/t}) \cdot t^2 \cdot (\beta \cdot \sqrt{r/t})} = 2^{\beta^2 \cdot t \cdot r}$$

and the upper bound on the number of messages in Fact 1.1 is met, provided β is sufficiently small. But then $|Q| = 2^{\Omega(t)}$ follows from (5.1) and hence $|Q| \geq 2^{\gamma \cdot t}$ holds for some positive constant γ. We obtain $2^{\gamma \cdot t} \leq |Q| < 2^{\beta \cdot \sqrt{r/t}}$ and hence $2^{\gamma \cdot t} < 2^{\beta \cdot \sqrt{r/t}} = 2^{\beta \cdot t}$, since $t = r^{1/3}$. We get a contradiction if β is chosen sufficiently small and we have shown

Lemma 5.2. *Let N be an NFA with sublinear ambiguity recognizing $\exists_1(L)$. Then N has at least $2^{\Omega(r^{1/3})}$ states.* ∎

6. A Hierarchy for Polynomial Ambiguity

Let $k \geq 1$ be arbitrary and let N be an NFA for $\exists_k(L)$. We again follow the strategy for $k = 1$, however the transition from NFA's to communication protocols is now more involved. For $k > 1$ we have to work with vectors $(p_0, p_1, \ldots, p_{2(k-1)}, p_{2(k-1)+1})$ of states and besides reachabilty for p_0 and acceptance by $p_{2(k-1)+1}$ we also have to guarantee that computation paths exist between p_{2i} and p_{2i+1}. This last requirement requires some further work.

Definition 6.1. Let $(\xi_0, \xi_1) \in \exists_{=0}(L) \times \exists_{=0}(L)$ be arbitrary. We say that the vector $(p_0, p_1, \ldots, p_{2(k-1)}, p_{2(k-1)+1})$ is critical for (ξ_0, ξ_1) iff

(1) all strings in $\exists_{=0}(L) \cdot \xi_0$ reach p_0 and all strings in $\xi_1 \cdot \exists_{=0}(L)$ are accepted by $p_{2(k-1)+1}$
(2) and for all strings $u \in \exists_{=0}(L)$ and for all i $(0 \leq i < k-1)$ there is a string v such that a computation for $\xi_1 uv$ starts in p_{2i+1} and ends in $p_{2(i+1)}$.

We construct ξ_0 as in Lemma 3.2 and hence for any state p of the NFA N either all strings in $\exists_{=0}(L) \cdot \xi_0$ reach p or no such string reaches p. To construct ξ_1 we first run the procedure of Lemma 3.2 and property (1) is satisfied. Then we process all pairs (p, q) of states of N in some arbitrary order. If for all strings $u \in \exists_{=0}(L)$ there is a string $v \in \exists_{=0}(L)$ such that $\xi_1 uv$ has a computation beginning in p and ending in q, then we say that the pair (p, q) is "alive" and ξ_1 is left unchanged. Otherwise there is a string $u \in \exists_{=0}(L)$ such that no computation for a string in $\xi_1 \cdot u \cdot \exists_{=0}(L)$ has a computation beginning in p and ending in q. We replace ξ_1 by $\xi_1 u$. The pair (p, q) is now "dead", since no string in $\xi_1 \cdot \exists_{=0}(L)$ has a computation beginning in p and ending in q. Also observe that processed pairs do not change their status, i.e., remain dead, resp. stay alive after updating ξ_1. We have generalized Lemma 3.2.

Lemma 6.2. *Let N be an NFA for $\exists_k(L)$. Then there are strings $\xi_0, \xi_1 \in \exists_{=0}(L)$ such that*

$$\bigcup_{(p_0,\ldots,p_{2k-1}) \text{ is critical for } (\xi_0,\xi_1)} \{z \in L \mid p_{2i+1} \xrightarrow{z} p_{2(i+1)} \text{ for all } 0 \leq i < k-1\} = L.$$

Proof. The argument is analogous to the proof of Lemma 3.2. This time we have to observe that accepting computations for strings in $\xi_0 \cdot (L \cdot \xi_1)^k$ have to traverse critical vectors. ∎

For $k = 1$ Lemma 3.3 establishes that a string $S(z)$ "lives" in a launching cycle for p_0 and a storage cycle for p_1. Its generalization requires more work. Let $\vec{p} = (p_0, \ldots, p_{2k-1})$ be a critical vector and let $z \in K \setminus L$ be an arbitrary string. We construct a string $u \in \exists_{=0}(L)$ for z so that some string with period uz can be launched by p_0, stored and launched in between p_{2i+1} and $p_{2(i+1)}$ and finally stored by $p_{2(k-1)+1}$. In particular, we say that a string $u \in \exists_{=0}(L)$ is appropriate for z if the following properties are satisfied:

(1) $(zu)^{|Q|}$ reaches p_0.
(2) For every i, $0 \leq i < k-1$, there is a string s_i and computations $p_{2i+1} \xrightarrow{s_i} p_{2(i+1)}$. Moreover, s_i starts with a suffix of uz containing ξ_1 as prefix, followed by $(uz)^{|Q|}$ and completed by a prefix of u.
(3) State $p_{2(k-1)+1}$ accepts any string s_{k-1} which consists of a suffix of uz containing ξ_1 as prefix, followed by $(uz)^{|Q|}$.
(4) The string $s = (zu)^{|Q|}zs_0z\cdots zs_{k-2}zs_{k-1}$ has periods zu and uz respectively.

Now assume that u is appropriate for z. We show that the string $S(z) = s$ "lives" in appropriate cycles for each p_i. First observe that $S(z)$ has period zu and hence also period uz. The proof of Lemma 3.3 shows that a launching cycle $r \xrightarrow{(zu)^a} r \xrightarrow{(zu)^{a_1}} p_0$ is established, once $(zu)^{|Q|}$ reaches p_0. Also, intermediate cycles in between p_{2i+1} and $p_{2(i+1)}$ exist, since s_i has substring $(uz)^{|Q|}$, and a final storage cycle following $p_{2(k-1)+1}$ exists, since $p_{2(k-1)+1}$ accepts a string with suffix $(uz)^{|Q|}$.

Lemma 6.3. *For every string $z \in K \setminus L$ there is an appropriate string $u \in \exists_{=0}(L)$ for z.*

Proof. Let q_l be some arbitrary ordering of the states of N. Each pair $(p_{2i+1}, p_{2(i+1)})$ influences the construction of u. Assume for the moment that strings $\xi_{i,l}$ are already defined. We set

$$u_{i,j} = \xi_1 \cdot \Pi_{l \leq j, (q_l, p_{2(i+1)}) \text{ is alive}} \; z\xi_1\xi_{i,l}$$

for all j $(1 \leq j \leq |Q|)$. Observe that $u_{i,j} = u_{i,j-1} \cdot (z\xi_1\xi_{i,j})$, if $(q_j, p_{2(i+1)})$ is alive, and that ξ_1 is a prefix of $u_{i,j}$. Choose the strings $\xi_{i,l} \in \exists_{=0}(L)$ so that there is a computation for $u_{i,j}$ from q_j to $p_{2(i+1)}$. Such strings $\xi_{i,l}$ exist with property (2) of a critical vector, since ξ_1 is a prefix of $u_{i,j}$ and $(q_j, p_{2(i+1)})$ is alive. Finally set

$$u_i = u_{i,|Q|} \cdot z\xi_1 \quad \text{and} \quad u = u_0 \cdots u_{k-2} \cdot \xi_0.$$

We show that u is appropriate for z by first verifying property (1). The string u has suffix ξ_0 and hence, by property (1) of a critical vector, $(zu)^{|Q|}$ reaches p_0, the first component of the critical vector $\vec{p}$.

Observe that each $u_{i,j}$ has prefix ξ_1 and hence u_i and u have ξ_1 as prefix. We start the verification of properties (2) and (3) by defining s_0 and constructing a computation $p_1 \xrightarrow{s_0} p_2$. Since ξ_1 is a prefix of u, there is a computation for $(uz)^{|Q|}$ which leads from p_1 to a state q_j such that the pair (q_j, p_2) is alive. But then, by definition of $u_{0,j}$, there is a computation for $(uz)^{|Q|} \cdot u_{0,j}$ which starts in p_1, reaches q_j after reading $(uz)^{|Q|}$ and ends in p_2 after reading $u_{0,j}$. We set $s_0 = (uz)^{|Q|} \cdot u_{0,j}$. By construction, $u_{0,j}$ is a prefix of u_0 which itself is a prefix of u. Thus there is a string $v_{0,j}$ with $u = u_{0,j} \cdot z \cdot v_{0,j}$ and $v_{0,j}$ has prefix ξ_1.

We now construct a string s_1 and a computation $p_3 \xrightarrow{s_1} p_4$ as follows. Since $v_{0,j}$ has prefix ξ_1 there is a computation for $v_{0,j} \cdot z \cdot (uz)^{|Q|} \cdot u_0$ which reaches a state q_k when starting in state p_3. Since the pair (q_k, p_4) is alive, we obtain the computation

$$p_3 \xrightarrow{v_{0,j} z (uz)^{|Q|} u_0} q_k \xrightarrow{u_{1,k}} p_4$$

and set $s_1 = v_{0,j}z(uz)^{|Q|}u_0u_{1,k}$. The construction of s_i and verifying a computation $p_{2i+1} \xrightarrow{s_i} p_{2(i+1)}$ for arbitrary $i < k-1$ proceeds in a completely analogous fashion. Finally, again by property (1) of a critical vector, state $p_{2(k-1)+1}$ accepts any string s_{k-1} consisting of a suffix of uz followed by $(zu)^{|Q|}$, since the suffix of uz has prefix ξ_1.

To complete the argument observe that by construction $s = (zu)^{|Q|} z s_0 z \cdots z s_{k-2} z s_{k-1}$ has periods uz and zu respectively. ∎

The remainder of the argument proceeds completely analogous to the case of $k = 1$. Lemma 3.4 shows that an NFA with sublinear ambiguity solves the detection problem for $k = 1$. To introduce its generalization we firstly introduce the detection problem for $k > 1$: z has to be rejected iff there is a string $z' \in K \setminus L$ such that z, acting as an impostor of z', can be placed in matching positions within the $k + 1$ individual uz'-cycles of N. Lemma 3.4 was a direct consequence of Lemma 3.3 in the case of $k = 1$. In the same manner we can now show that an NFA with ambiguity $o(n^k)$ solves the detection problem for general k as a direct consequence of Lemma 6.3.

Let N be an NFA with ambiguity $o(n^k)$ for $\exists_k(L)$. As in Lemma 4.1 we simulate N to obtain a nondeterministic protocol P solving the detection problem with $|Q|^{O(kt)}$ messages; the exponent grows by the factor k, since $k+1$ instead of two computations of N on input z have to be simulated. We transform P into a deterministic protocol D with $|Q|^{O((kt)^2 \log |Q|)}$ messages as in Lemma 5.1. To complete the proof of Theorem 1.2, we replace r by r/k^2 in the proof of Lemma 5.2 (to compensate for the increase in the number of messages of D from $|Q|^{O(t^2 \log |Q|)}$ to $|Q|^{O(k^2 t^2 \log |Q|)}$) and obtain

Lemma 6.4. *Let N be an NFA with ambiguity $o(n^k)$ recognizing $\exists_k(L)$. Then N has at least $2^{\Omega((r/k^2)^{1/3})}$ states.* ∎

References

[1] Aho, A.V., Ullman, J.D. and Yannakakis, M., On notions of information transfer in VLSI circuits, *Proc. of the 15th Annual STOC*, pp. 133-139, 1983.

[2] Hromkovič, J., Karhumäki, J., Klauck, H., and Schnitger, G., Communication complexity method for measuring nondeterminism in finite automata, *Inf. Comput.* 172, pp. 202-217, 2002.

[3] Hromkovič, J., and Schnitger, G., Nondeterministic Communication with a Limited Number of Advice Bits, *SIAM J. Comput.* 33(1), pp. 43-68, 2003.

[4] Leung, H., Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata, *SIAM. J. Comput.* 27, pp. 1073-1082, 1998.

[5] Lovasz, L., Communication complexity: a survey, in "Paths, Flows and VLSI Layout", Korte, Lovasz, Prömel, Schrijver eds., Springer Verlag, pp. 235-266, 1990.

[6] Ravikumar, B., and Ibarra, O., Relating the type of ambiguity of finite automata to the succinctness of their representation, *SIAM J. Comput.* 19, pp. 1263-1282, 1989.

[7] R.E. Stearns and H.B. Hunt III, On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata, *SIAM J. Comput.* 14(3), pp. 598-611, 1985.

[8] A. Weber and H. Seidl, On the degree of ambiguity of finite automata, *Theor. Comput. Sci.* 88 (2), pp. 325-349, 1991.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 565–576
www.stacs-conf.org

ON THE BOREL INSEPARABILITY OF GAME TREE LANGUAGES

SZCZEPAN HUMMEL [1] AND HENRYK MICHALEWSKI [1] AND DAMIAN NIWIŃSKI [1]

[1] Faculty of Mathematics, Informatics, and Mechanics
Warsaw University, Poland
E-mail address: {shummel,henrykm,niwinski}@mimuw.edu.pl

Abstract. The game tree languages can be viewed as an automata-theoretic counterpart of parity games on graphs. They witness the strictness of the index hierarchy of alternating tree automata, as well as the fixed-point hierarchy over binary trees.

We consider a game tree language of the first non-trivial level, where Eve can force that 0 repeats from some moment on, and its dual, where Adam can force that 1 repeats from some moment on. Both these sets (which amount to one up to an obvious renaming) are complete in the class of co-analytic sets. We show that they cannot be separated by any Borel set, hence *a fortiori* by any weakly definable set of trees.

This settles a case left open by L.Santocanale and A.Arnold, who have thoroughly investigated the separation property within the μ-calculus and the automata index hierarchies. They showed that separability fails in general for non-deterministic automata of type Σ^μ_n, starting from level $n = 3$, while our result settles the missing case $n = 2$.

Introduction

In 1970 Rabin [15] proved the following property: If a set of infinite trees can be defined both by an *existential* and by a *universal* sentence of monadic second order logic then it can also be defined in a weaker logic, with quantification restricted to *finite* sets. An automata-theoretic counterpart of this fact [15, 12] states that if a tree language, as well as its complement, are both recognizable by Büchi automata (called *special* in [15]) then they are also recognizable by weak alternating automata. Yet another formulation, in terms of the μ-calculus [3], states that if a tree language is definable both by a Π^μ_2-term (i.e., with a pattern $\nu\mu$) and a Σ^μ_2-term ($\mu\nu$), then it is also definable by an alternation free term, i.e., one in $\mathit{Comp}(\Pi^\mu_1 \cup \Sigma^\mu_1)$. This last formulation gives rise to a question if the equation

$$\Pi^\mu_n \cap \Sigma^\mu_n = \mathit{Comp}(\Pi^\mu_{n-1} \cup \Sigma^\mu_{n-1})$$

holds on all levels of the fixed-point hierarchy. Santocanale and Arnold showed [17], rather surprisingly, that it is not the case for $n \geq 3$. They exhibit a series of "ambiguous" properties, expressible by terms in Π^μ_n and in Σ^μ_n, but not in $\mathit{Comp}(\Pi^\mu_{n-1} \cup \Sigma^\mu_{n-1})$. On positive side

1998 ACM Subject Classification: F.1.1 Automata, F.4.1 Set theory, F.4.3 Classes defined by grammars or automata.

Key words and phrases: Tree automata, Separation property, Borel sets, Parity games.

All authors are supported by Grant N206 008 32/0810.

however, they discover a more subtle generalization of Rabin's result, which continues to hold on the higher stages of the hierarchy.

Let us explain it at a more abstract level, with $\mathcal{L}$ ("large") and $\mathcal{S}$ ("small") being two classes of subsets of some universe U. Consider the following properties.

> *Simplification.* Whenever L and its complement $\bar{L}$ are both in $\mathcal{L}$, they are also in $\mathcal{S}$.
>
> *Separation.* Any two disjoint sets $L, M \in \mathcal{L}$ are separated by some set K in $\mathcal{S}$ (i.e., $L \subseteq K \subseteq U - M$).

Note that (given some $\mathcal{L}$ and $\mathcal{S}$) separation implies simplification, but in general not *vice versa*. In topology, it is well known (see, e.g., [11]) that the separation property holds for

$\mathcal{L}$ = analytic ($\mathbf{\Sigma}^1_1$) subsets of a Polish space (e.g., $\{0,1\}^\omega$),
$\mathcal{S}$ = Borel sets,

but fails for $\mathcal{L}$ = co-analytic sets ($\mathbf{\Pi}^1_1$) and $\mathcal{S}$ as above. On the other hand both classes enjoy the simplification property (which amounts to the Suslin Theorem).

In this setting, Rabin's result establishes the simplification property for

$\mathcal{L}$ = Büchi definable tree languages (Π^μ_2 in the fixed-point hierarchy),
$\mathcal{S}$ = weakly definable tree languages ($Comp(\Pi^\mu_1 \cup \Sigma^\mu_1)$).

A closer look at the original proof reveals that a (stronger) separation property also holds for these classes.

Santocanale and Arnold [17] showed in turn that the separation property holds for

$\mathcal{L}$ = tree languages recognizable by *non-deterministic* automata of level Π^μ_n,
$\mathcal{S}$ = tree languages definable by fixed-point terms in $Comp(\Pi^\mu_{n-1} \cup \Sigma^\mu_{n-1})$,

for the remaining case of $n \geq 3$. On the negative side, they showed that the separation property fails for $\mathcal{L}$ consisting of tree languages recognizable by *non-deterministic* automata of level Σ^μ_n, for $n \geq 3$, leaving open the case of $n = 2$. In fact, their proof reveals that, in the case under consideration, even a (weaker) simplification property fails (see [17], section 2.2.3). As for Σ^μ_2 however, the simplification property does hold, because of Rabin's result[1]. For this reason, the argument of Santocanale and Arnold cannot be extended to the class Σ^μ_2. In the present paper, we show that the separation property fails also in this case, completing the missing point in the classification of [17].

We use a topological argument and show in fact a somewhat stronger result, exhibiting two disjoint languages recognized by non-deterministic tree automata with co-Büchi condition (i.e., Σ^μ_2), which cannot be separated by any *Borel* set (in a standard Cantor-like topology on trees). The languages in question are the so-called *game tree languages* (of level (0,1)), which were used in [8] (and later also in [2]) in the proof of the strictness of the fixed-point hierarchy over binary trees. More specifically, one of these languages consists of the trees labeled in $\{0,1\} \times \{\exists, \forall\}$, such that in the induced game (see definition below) *Eve* has a strategy to force only 0's from some moment on. The second is the twin copy of the first and consists of those trees that *Adam* has a strategy to force only 1's from some moment on.

[1] If a set and its complement are recognized by non-deterministic co-Büchi automata then they are also both recognized by alternating Büchi automata [5], and hence by non-deterministic Büchi automata, and hence are weakly definable [15].

The wording introduced above differs slightly from the standard terminology of descriptive set theory, where a *separation property* of a class $\mathcal{L}$ means our property with $\mathcal{S} = \{X : X, \bar{X} \in \mathcal{L}\}$ (see [11]). To emphasize the distinction, following [1], we will refer to the latter as to the *first separation property*. In this setting, the first separation property holds for the class of Büchi recognizable tree languages, but it fails for the co-Büchi languages, similarly as it is the case of the analytic *vs.* co-analytic sets, mentioned above. This may be read as an evidence of a strong analogy between the Büchi class and $\mathbf{\Sigma}^1_1$. In fact, Rabin [15] early observed that the Büchi tree languages are definable by existential sentences of monadic logic, and hence analytic. We show however that, maybe surprisingly, the converse is not true, by exhibiting an analytic tree language, recognized by a parity (Rabin) automaton, but not by any Büchi automaton.

Note. The fixed-point hierarchy discussed above provides an obvious context of our results, but in the paper we do not rely on the μ-calculus concepts or methods. For definitions of relevant concepts, we refer an interested reader to the work by Santocanale and Arnold [17] or, e.g., to [4].

1. Basic concepts

Throughout the paper, ω stands for the set of natural numbers.

Metrics on trees. A full binary tree over a finite alphabet Σ (or shortly a tree, if confusion does not arise) is represented as a mapping $t : \{1,2\}^* \to \Sigma$.

We consider the classical topology *à la Cantor* on T_Σ induced by the metric

$$d(t_1,t_2) = \begin{cases} 0 & \text{if } t_1 = t_2 \\ 2^{-n} \text{ with } n = \min\{|w| : t_1(w) \neq t_2(w)\} & \text{otherwise} \end{cases} \tag{1.1}$$

It is well-known and easy to see that if Σ has at least two elements then T_Σ with this topology is homeomorphic to the Cantor discontinuum $\{0,1\}^\omega$. Indeed, it is enough to fix a bijection $\alpha : \omega \to \{1,2\}^*$, and a mapping (code) $C : \Sigma \to \{0,1\}^*$, such that $C(\Sigma)$ forms a maximal antichain w.r.t. the prefix ordering. Then $T_\Sigma \ni t \mapsto C \circ t \circ \alpha \in \{0,1\}^\omega$ is a desired homeomorphism. We assume that the reader is familiar with the basic concepts of set-theoretic topology (see, e.g., [11]). The *Borel sets* over T_Σ constitute the least family containing open sets and closed under complement and countable union. The *Borel relations* are defined similarly, starting with open relations (i.e., open subsets of T^n_Σ, for some n, considered with product topology). The *analytic* (or $\mathbf{\Sigma}^1_1$) sets are those representable by

$$L = \{t : (\exists t')\, R(t,t')\}$$

where $R \subseteq T_\Sigma \times T_\Sigma$ is a Borel relation. The *co-analytic* (or $\mathbf{\Pi}^1_1$) sets are the complements of analytic sets. A continuous mapping $f : T_\Sigma \to T_\Sigma$ *reduces* a tree language $A \subseteq T_\Sigma$ to $B \subseteq T_\Sigma$ if $f^{-1}(B) = A$. As in complexity theory, a set $L \in \mathcal{K}$ is *complete* in class $\mathcal{K}$ if all sets in this class reduce to it.

Non-deterministic automata. A non-deterministic tree automaton over trees in T_Σ with a parity acceptance condition[2] is presented as $A = \langle \Sigma, Q, q_I, Tr, rank\rangle$, where Q is a finite set of *states* with an *initial state* q_I, $Tr \subseteq Q\times\Sigma\times Q\times Q$ is a set of *transitions*, and $rank : Q \to \omega$ is the *ranking* function. A transition (q,σ,p_1,p_2) is usually written $q \xrightarrow{\sigma} p_1,p_2$.

A *run* of A on a tree $t \in T_\Sigma$ is itself a Q–valued tree $\rho : \{1,2\}^* \to Q$ such that $\rho(\varepsilon) = q_I$, and, for each $w \in dom\,(\rho)$, $\rho(w) \xrightarrow{t(w)} \rho(w1),\rho(w2)$ is a transition in Tr. A *path* $P = p_0p_1\ldots \in \{1,2\}^\omega$ in ρ is *accepting* if the highest rank occurring infinitely often along it is even, i.e., $\limsup_{n\to\infty} rank(\rho(p_0p_1\ldots p_n))$ is even. A *run is accepting* if so are all its paths. A tree language $T(A)$ *recognized* by A consists of those trees in T_Σ which admit an accepting run.

The *Rabin–Mostowski index* of an automaton A is the pair $(\min(rank(Q)), \max(rank(Q)))$; without loss of generality, we may assume that $\min(rank(Q)) \in \{0,1\}$.

An automaton with the *Rabin–Mostowski index* $(1,2)$ is called a *Büchi automaton*. Note that a Büchi automaton accepts a tree t if, on each path, some state of rank 2 occurs infinitely often. We refer to the tree languages recognizable by Büchi automata as to *Büchi (tree) languages*. The *co-Büchi languages* are the complements of Büchi languages. It is known that if a tree language is recognized by a non-deterministic automaton of index $(0,1)$ then it is co-Büchi[3]; the converse is not true in general (see the languages $M_{i,k}$ in Example 1.1 below).

Example 1.1. Let

$$L = \{t \in T_{\{0,1\}} : (\exists P)\ \limsup_{n\to\infty} t(p_0p_1\ldots p_n) = 1\}$$

This set is recognized by a Büchi automaton with transitions

$$q/p \xrightarrow{0} q,T; \qquad q/p \xrightarrow{1} p,T; \qquad T \xrightarrow{(0/1)} T,T;$$
$$q/p \xrightarrow{0} T,q; \qquad q/p \xrightarrow{1} T,p;$$

with $rank(q) = 1$ and $rank(p) = rank(T) = 2$. Rabin [15] showed that its complement $\bar{L}$ cannot be recognized by any Büchi automaton, but it is recognizable by an (even deterministic) automaton of index $(0,1)$

$$0/1 \xrightarrow{0} 0,0; \qquad 0/1 \xrightarrow{1} 1,1; \qquad rank(i) = i; \quad for \quad i = 0,1.$$

This last set can be generalized to the so-called parity languages (with $i \in \{0,1\}$)

$$M_{i,k} = \{t \in T_{\{i,\ldots,k\}} : (\forall P)\ \limsup_{n\to\infty} t(p_0p_1\ldots p_n)\ \ is\ \ even\}$$

which are all co-Büchi but require arbitrary high indices [13]. It can also be showed that all languages $M_{i,k}$ (except for $(i,k) = (0,0),(1,1),(1,2)$) are complete in the class of co-analytic sets $\mathbf{\Pi}^1_1$ (see, e.g., [14]).

The class of languages which are simultaneously Büchi and co-Büchi has numerous characterizations mentioned in the introduction; all these characterizations easily imply that such sets are Borel (even of finite Borel rank).

[2]Currently most frequently used in the literature, these automata are well-known to be equivalent to historically previous automata with the Muller or Rabin conditions [18].

[3]It follows, in particular, from the equivalence of the non-deterministic and alternating Büchi automata [5], mentioned in footnote 1.

Example 1.2. Consider the set $\bar{L} = M_{0,1}$ of Example 1.1, and its twin copy obtained by the renaming $0 \leftrightarrow 1$,

$$M'_{0,1} = \{t \in T_{\{0,1\}} : (\forall P) \liminf_{n\to\infty} t(p_0 p_1 \dots p_n) = 1\}.$$

The sets $M_{0,1}$ and $M'_{0,1}$ are disjoint, co-Büchi and, as we have already noted, $\mathbf{\Pi}^1_1$ complete. They can be separated by a set K of trees[4], such that on the *rightmost* branch, there are only finitely many 1's

$$K = \{t \in T_{\{0,1\}} : \limsup_{n\to\infty} t(\underbrace{22\dots2}_{n}) = 0\}$$

(i.e., $M_{0,1} \subseteq K \subseteq T_{\{0,1\}} - M'_{0,1}$). The set K can be presented as a countable union of closed sets

$$K = \bigcup_m \{t \in T_{\{0,1\}} : (\forall n \geq m)\, t(\underbrace{22\dots2}_{n}) = 0\}$$

so it is on the level $\mathbf{\Sigma}^0_2$ (i.e., F_σ) of the Borel hierarchy. The membership in the Borel hierarchy can also be seen trough an automata-theoretic argument by showing that K is simultaneously Büchi and co-Büchi. Indeed it can be recognized by an (even deterministic) automaton with co-Büchi condition

$$0/1 \xrightarrow{0} T, 0; \qquad 0/1 \xrightarrow{1} T, 1; \qquad T \xrightarrow{(0/1)} T, T; \qquad \mathit{rank}(i) = i, \ \mathit{for} \ \ i = 0, 1, \quad \mathit{rank}(T) = 0,$$

as well as by a (non-deterministic) Büchi automaton

$$q \xrightarrow{(0/1)} T, q/p; \qquad p \xrightarrow{0} T, p; \qquad T \xrightarrow{(0/1)} T, T; \qquad \mathit{rank}(q) = 1, \qquad \mathit{rank}(p) = \mathit{rank}(T) = 2.$$

We will see in the next section that a Borel separation of co-Büchi languages is not always possible.

2. Inseparable pair

Let

$$\Sigma = \{\exists, \forall\} \times \{0, 1\},$$

we denote by π_i the projection on the ith component of Σ. With each $t \in T_\Sigma$, we associate a game $G(t)$, played by two players, *Eve* and *Adam*. The positions of Eve are those nodes v, for which $\pi_1(t(v)) = \exists$, the remaining nodes are positions of Adam. For each position v, it is possible to move to one of its successors, $v1$ or $v2$. The players start in the root and then move down the tree, thus forming an infinite path $P = (p_0 p_1 p_2 \dots)$. The successor is selected by Eve or Adam depending on who is the owner of the position $p_0 p_1 \dots p_{n-1}$. The play is won by Eve if

$$\limsup_{n\to\infty} \pi_2\left(t(p_0 p_1 \dots p_n)\right) = 0$$

i.e., 1 occurs only finitely often, otherwise Adam is the winner. A strategy for Eve selects a move for each of her positions; it is winning if any play consistent with the strategy is won by Eve. We say that Eve *wins* the game $G(t)$ if she has a winning strategy. The analogous concepts for Adam are defined similarly.

A reader familiar with the *parity games* ([10], see also [18]) has noticed of course that the games $G(t)$ are a special case of these (with the index $(0, 1)$).

[4]This argument is due to Paweł Milewski.

Now let

$$W_{0,1} = \{t : \textit{Eve wins } G(t)\}$$

We also define a set $W'_{0,1} \subseteq T_\Sigma - W_{0,1}$, consisting of those trees t, where Adam has a strategy which guarantees him not only to win in $G(t)$, but also to force a stronger condition, namely

$$\liminf_{n\to\infty} \pi_2\left(t(p_0 p_1 \dots p_n)\right) = 1.$$

It should be clear that $W'_{0,1}$ can be obtained from $W_{0,1}$ by applying (independently on each component) a renaming $0 \leftrightarrow 1$, $\exists \leftrightarrow \forall$. Thus, the sets $W_{0,1}$ and $W'_{0,1}$ are disjoint, but have identical topological and automata-theoretic properties.

Let us see that the set $W_{0,1}$ can be recognized by a non-deterministic automaton of index $(0,1)$; it is enough to take the states $\{0,1\} \cup \{T\}$, with $\mathit{rank}(T) = 0$, and $\mathit{rank}(\ell) = \ell$, for $\ell \in \{0,1\}$, the initial state 0, and transitions

$$\ell \overset{(\forall,m)}{\to} m, m; \qquad \ell \overset{(\exists,m)}{\to} m, T; \qquad \ell \overset{(\exists,m)}{\to} T, m; \qquad T \overset{(Q,m)}{\to} T, T,$$

with $m \in \{0,1\}$, and $Q \in \{\exists, \forall\}$. Hence, the sets $W_{0,1}$ and $W'_{0,1}$ are co-Büchi (c.f. the remark before Example 1.1).

We are ready to state the main result of this paper.

Theorem 2.1. *The sets $W_{0,1}$ and $W'_{0,1}$ cannot be separated by any Borel set.*

Proof. The proof relies on the following.

Lemma 2.2. *For any Borel set $B \subseteq T_\Sigma$, there is a continuous function $f_B : T_\Sigma \to T_\Sigma$, such that*

$$\begin{array}{rcl} u \in B & \Rightarrow & f_B(u) \in W_{0,1} \\ u \notin B & \Rightarrow & f_B(u) \in W'_{0,1} \end{array}$$

Proof. Note that f_B is required to reduce simultaneously B to $W_{0,1}$ and $T_\Sigma - B$ to $W'_{0,1}$. We proceed by induction on the complexity of the set B.

Note first that if B is clopen (simultaneously closed and open) then it is enough to fix two trees $t \in W_{0,1}$ and $t' \in W'_{0,1}$, and define f_B by

$$\begin{array}{rcl} u \in B & \Rightarrow & f_B(u) = t \\ u \notin B & \Rightarrow & f_B(u) = t' \end{array}$$

Also note that, by symmetry of the sets $W_{0,1}$ and $W'_{0,1}$, the claim for B readily implies the claim for the complement $T_\Sigma - B$. (Specifically, $f_{B'}$ is obtained by composing f_B with a suitable renaming.)

Finally note that the space $T_\Sigma \approx \{0,1\}^\omega$ has a countable basis consisting of clopen sets.

Then, in order to complete the proof, it remains to settle the induction step for $B = \bigcup_{n<\omega} B_n$. Assume that we have already the reductions f_{B_n} satisfying the claim, for $n < \omega$. Given $u \in T_\Sigma$, we construct a tree $f_B(u)$, by labeling the rightmost path by $(\exists, 1)$, and letting a subtree in the node $2^n 1$ be $f_{B_n}(u)$ (see Figure 1). In symbols,

$$\begin{array}{rcl} f_B(u)(2^n) & = & (\exists, 1) \\ f_B(u)(2^n 1 v) & = & f_{B_n}(u)(v), \quad \textit{for } n < \omega, \quad v \in \{1,2\}^* . \end{array}$$

Since all the functions f_{B_n} are continuous, the resulting f_B is continuous as well. Now, if $u \in B_m$, for some m, then Eve has an obvious winning strategy: follow the rightmost path and turn left in 2^m, then use the winning strategy on the subtree $f_{B_m}(u)$, which exists, by induction hypothesis.

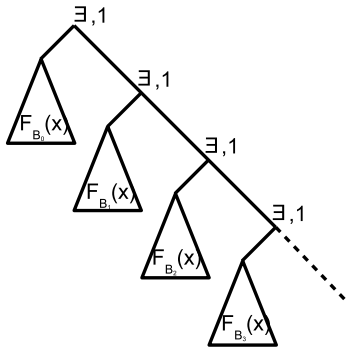


Figure 1: Induction step for $\bigcup_n B_n$.

If, however, $(\forall n)\, u \notin B_n$ then Adam can win the game with the stronger winning criterion, required in the definition of $W'_{0,1}$. Indeed, he can do so as soon as Eve enters any of the subtrees $f_{B_n}(u)$ (by induction hypothesis), but he also wins if Eve remains forever on the rightmost path.

This proves the claim for f_B, and thus completes the proof of the lemma. ∎

We are ready to complete the proof of the theorem. Suppose that there is a Borel set C, such that $W_{0,1} \subseteq C \subseteq T_\Sigma - W'_{0,1}$. The claim of the lemma immediately implies that

$$\begin{array}{rcl} u \in B & \Rightarrow & f_B(u) \in C \\ u \notin B & \Rightarrow & f_B(u) \in T_\Sigma - C. \end{array}$$

Thus any Borel set B over T_Σ is reducible to C, but this is clearly impossible, as it would contradict the strictness of the Borel rank hierarchy in the Cantor discontinuum $\{0,1\}^\omega$ (see, e.g., [11]). ∎

Since the sets $W_{0,1}$ and $W'_{0,1}$ are recognizable by non-deterministic automata of index $(0,1)$, Theorem 2.1 settles the case of $n = 2$, missing in Section 2.2.3 of [17], devoted to the failure of separation property for non-deterministic automata of type Σ^μ_n and the class $\mathit{Comp}(\Pi^\mu_{n-1} \cup \Sigma^\mu_{n-1})$.

In the terminology introduced at the end of introduction, we can state the following.

Corollary 2.3. *The class of co-Büchi tree languages does not have the first separation property.* ∎

This may be contrasted with the positive result of [15]. As we have mentioned in the introduction, Rabin's original proof essentially shows this property for the class of Büchi tree languages, although it is not explicitly stated there. For the sake of completeness, we sketch the argument below, following closely the μ-calculus version of [3] (based on the original proof of [15]).

Theorem 2.4 (Rabin)**.** *The class of Büchi tree languages has the first separation property.*

Proof. Let A and B be two non-deterministic Büchi automata, such that $T(A) \cap T(B) = \emptyset$. We will refer to the states of rank 2 as to *accepting* states (of the corresponding automaton). A *cut* (of a tree) is a finite maximal antichain in $\{1,2\}^*$ with respect to the prefix ordering $\leq$. For two cuts X, Y we let $Y > X$ if Y lies below X, i.e., $(\forall y \in Y)\,(\exists x \in X)\, y > x$. It is easy to see that a run ρ of a Büchi automaton is accepting if, for each cut X, there is a

cut $Y > X$, labeled by the accepting states (i.e., $(\forall y \in Y)\, rank(\rho(y)) = 2$). We inductively define a sequence of tree languages K_q^n, for each state q of A, and $n \geq 0$.

The set K_q^0 consists of all trees t which admit some run (not necessarily accepting) of A starting from q (q-run, for short). The set K_q^{n+1} comprises those trees t, which admit a q-run ρ, such that, for each cut X, there exists a cut $X' > X$, and a run ρ', with the following properties:

- ρ' agrees with ρ until the cut X,
- all states in $\rho'(X')$ are accepting,
- $(\forall v \in X')$, the subtree of t rooted in v (in symbols $t.v$) belongs to K_p^n, where $p = \rho'(v)$.

It follows by induction on n that $T(A) \subseteq K_{q_I}^n$, where q_I is the initial state of A. Now let n_A and n_B be the numbers of states of A and B, respectively, and let $M = 2^{n_A \cdot n_B} + 1$. We claim that $K_{q_I}^M$ separates $T(A)$ and $T(B)$. We already know that $T(A) \subseteq K_{q_I}^M$. For the sake of contradiction, suppose that $t \in K_{q_I}^M \cap T(B)$, and let ρ' be an accepting run of B on t.

Using the inductive definition of $K_{q_I}^M$, we can construct a sequence of cuts $X_1 < X_1' < \ldots < X_M < X_M'$, and a run ρ of A on t, such that

- $(\forall i \leq M)$ all states in $\rho(X_i)$ are accepting,
- $(\forall i \leq M,\, \forall v \in X_i)\, t.v \in K_{\rho(v)}^{M-i}$,
- $(\forall i \leq M)$ all states in $\rho'(X_i')$ are accepting.

By the choice of M, there exist $1 \leq k < \ell \leq M$, such that

$$\{(\rho(u), \rho'(u)) : u \in X_k\} = \{(\rho(v), \rho'(v)) : v \in X_\ell\}$$

Note that, by construction,

$$X_k < X_k' < X_l$$

with all states in $\rho'(X_k')$ accepting. Hence, by a standard tree-pumping argument, we can construct a new tree along with two accepting runs: by A and by B, contradicting $T(A) \cap T(B) = \emptyset$.

It remains to show that the language $K_{q_I}^M$ is both Büchi and co-Büchi. A direct construction of two Büchi automata would be somewhat cumbersome, but one can use here any of the characterizations of this intersection class mentioned above. In the proof given in [3], it is shown that the sets K_q^n are definable in the alternation-free μ-calculus. A reader familiar with monadic second-order logic can easily see that these languages are definable in its *weak* fragment, i.e., with quantifiers restricted to finite sets. This is enough as well, according to the characterization given by Rabin [15]. ∎

3. Broken analogy

A reader familiar with descriptive set theory may think of another inseparable pair of recognizable tree languages, induced by a classical example ([11], section 33.A). We will explain why it would not be useful for our purpose. Let us now consider non-labeled trees, i.e., subsets $T \subseteq \omega^*$ closed under initial segments. They can be viewed as elements of the Cantor discontinuum $\{0,1\}^\omega$ by fixing a bijection $\iota : \omega \to \omega^*$ and identifying a tree T with its characteristic function, given by $f_T(n) = 1$ iff $\iota(n) \in T$. In particular, we can discuss

topological properties of sets of such trees. As before, $P \in \omega^\omega$ is a path in a tree T if all finite prefixes of P are in T. Let

$$\begin{aligned} \mathrm{WF} &= \{T : T \text{ has no infinite path } \} \\ \mathrm{UB} &= \{T : T \text{ has } \textit{exactly one} \text{ infinite path } \} \end{aligned}$$

Both sets are known to be $\mathbf{\Pi}^1_1$-complete, although the membership of UB in $\mathbf{\Pi}^1_1$ is not obvious, and is the subject of one of Lusin's theorems (Theorem 18.11 in [11]). WF and UB are also known to be inseparable by Borel sets ([11], section 35, see also [6]). Now, it is not difficult to "encode" these sets as languages of labeled binary trees, which turn out to be recognizable by parity automata. In [14] a continuous reduction of WF to $M_{0,1}$ was used to show that the latter set is complete in $\mathbf{\Pi}^1_1$ (Example 1.1 above). Let

$$\begin{aligned} \mathrm{UB}_{bin} = \{t \in T_{\{0,1\}} : \quad & \text{there is } \textbf{exactly one} \text{ path } P \\ & \text{with } \limsup_{n\to\infty} t(p_0 p_1 \dots p_n) = 1\} \end{aligned}$$

It is easy to construct a non-deterministic automaton accepting this language; one can also assure that this automaton is non-ambiguous, i.e., for each accepted tree, has exactly one accepting run. From considerations above, one can deduce that the sets $T_{0,1}$ and UB_{bin} are inseparable by Borel languages. However, the language UB_{bin} is not co-Büchi.

Proposition 3.1. *The language $\overline{\mathrm{UB}_{bin}}$ is recognizable and analytic, but not Büchi.*

Proof. Let us call a path with infinitely many 1's *bad*. So the above language consists of trees that have either none or at least two bad paths. Rabin [15] shows that the language $T_{0,1}$ (no bad paths) cannot be recognized by a Büchi automaton, by constructing a correct tree which by pumping argument can be transformed to a tree with *exactly one* bad path (mistakingly accepted by the hypothetical automaton). So this classical argument applies to the language UB_{bin} without any changes. ∎

As we have argued in the introduction, this example somehow breaks the analogy between the class of Büchi recognizable tree languages and that of analytic sets. It turns out that the topological complexity, and the automata-theoretic complexity, although closely related, do not always coincide.

4. Conclusion

The automata-theoretic hierarchies, in particular the index hierarchies for non-deterministic and alternating tree automata, are studied because of the issues of expressibility and complexity. Typically, the higher the level in the hierarchy, the higher the expressive power of automata, but also the complexity of the related algorithmic problems (like emptiness or inclusion). Once the strictness of the hierarchy is established [7, 8], the next important problem is an *effective simplification*, i.e., determining the exact level of an object (e.g., a tree language) in the hierarchy. The problem is generally unsolved (see [9] for a recent development in this direction). One may expect that a better understanding of structural properties of the hierarchy can bring a progress also in this problem. We believe that ideas coming from descriptive set theory, like separation and reduction properties, uniformization, or completeness, can be helpful here.

The inseparable pairs of co-analytic sets are common in mathematics. Natural examples include the set of all continuous real–valued functions on the unit interval $[0, 1]$ which are everywhere differentiable together with the set of all continuous real–valued functions on

the unit interval $[0,1]$ which are not differentiable in exactly one point, but as in this case, other examples usually reflect the same pattern of WF *vs.* UB (c.f. [6]). In contrast, our pair presented in Section 2 is very symmetric: the two sets are copies of each other up to a symbolic renaming. Recently, Saint Raymond [16] established that the pair WF *vs.* UB is complete (in the sense of Wadge) with respect to all coanalytic pairs in the Cantor set. In the proof he uses an interesting example of another complete coanalytic pair, which exhibits certain symmetric properties. Building on his results, in subsequent work, we show that the pair $W_{0,1}$, $W'_{0,1}$, has an analogous completeness property.

Our example shows that the first separation property fails for the co-Büchi class (Σ^μ_2 in the fixed-point hierarchy) while, by Rabin results [15], it holds for the Büchi class (Π^μ_2). By this we have also settled a missing case in a classification by Santocanale and Arnold [17]. However, these authors were interested in the *relative* separation property (as explained in our introduction), as they primarily wanted to find if the *ambiguous class* $\Pi^\mu_n \cap \Sigma^\mu_n$ can be effectively captured by $Comp(\Pi^\mu_{n-1} \cup \Sigma^\mu_{n-1})$. As this coincidence turned out to fail for $n \geq 3$, it is meaningful to ask if the status of the first separation property established for the Büchi/co-Büchi classes, continues to hold for the higher-level classes Π^μ_n/Σ^μ_n. That is, if two disjoint sets definable in Π^μ_n can always be separated by a set in $\Pi^\mu_n \cap \Sigma^\mu_n$. (A similar question for Σ^μ_n, with expected answer negative.) In our opinion, it is an interesting problem, which may challenge for a better understanding of the topological structure of recognizable languages above $\mathbf{\Pi}^1_1 \cup \mathbf{\Sigma}^1_1$.

References

[1] J. W. Addison. Tarski's theory of definability: common themes in descriptive set theory, recursive function theory, classical pure logic, and finite-universe logic. *Annals of Pure and Applied Logic*, 126 (2004), 77–92.

[2] A. Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *RAIRO-Theoretical Informatics and Applications*, 33 (1999), 329–339.

[3] A. Arnold and D. Niwiński. Fixed point characterization of weak monadic logic definable sets of trees. In M.Nivat, A.Podelski, editors, *Tree Automata and Languages*, Elsevier, 1992, 159-188.

[4] A. Arnold and D. Niwiński. Rudiments of μ-Calculus. Elsevier Science, Studies in Logic and the Foundations of Mathematics, 146, North–Holland, Amsterdam, 2001.

[5] A. Arnold and D. Niwiński. Fixed point characterization of Büchi automata on infinite trees. J. Inf. Process. Cybern. EIK, 26, 1990, 453–461.

[6] H. Becker. Some Examples of Borel Inseparable Pairs of Co-analytic Sets. Mathematika 33, 1986, 72–79.

[7] J. C. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoret. Comput. Sci.*, 195 (1997), 133–153.

[8] J. C. Bradfield. Simplifying the modal mu-calculus alternation hierarchy. In: *Proc. STACS'98*, Lect. Notes Comput. Sci. 1373 (1998), 39–49.

[9] T. Colcombet and C. Löding. The Non-deterministic Mostowski Hierarchy and Distance-Parity Automata. *Proc. ICALP 2008*, Lect. Notes Comput. Sci. 5126 (2008), 398-409.

[10] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In: *Proceedings 32th Annual IEEE Symp. on Foundations of Comput. Sci.* (1991), 368–377.

[11] A. S. Kechris. Classical descriptive set theory. Springer-Verlag, New York, 1995.

[12] D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating Automata, the Weak Monadic Theory of Trees and its Complexity. *Theoret. Comput. Sci.* 97(2), (1992), 233-244.

[13] D. Niwiński. On fixed point clones. In: ICALP'86, Lect. Notes Comput. Sci. 226, Springer-Verlag, 1986, 464–473.

[14] D. Niwiński and I. Walukiewicz. A gap property of deterministic tree languages. *Theoret. Comput. Sci.* 303 (2003), 215–231.

[15] M. O. Rabin. Weakly definable relations and special automata. In: *Mathematical Logic and Foundations of Set Theory*, Y. Bar-Hillel ed., 1970, 1-23.
[16] J. Saint Raymond. Complete pairs of coanalytic sets. *Fundamenta Mathematicae* 194 (2007), 267–281.
[17] L. Santocanale and A. Arnold. Ambiguous classes in μ-calculi hierarchies. *Theoret. Comput. Sci.* 333 (2005), 265-296.
[18] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Springer-Verlag, 1997, pp. 389–455.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 577–588
www.stacs-conf.org

EQUATIONS OVER SETS OF NATURAL NUMBERS WITH ADDITION ONLY

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 of the form $X = YZ$ and $X = C$ are considered, in which the unknowns are sets of natural numbers, "+" denotes pairwise sum of sets $S+T = \{m+n \mid m \in S,\ n \in T\}$, and C is an ultimately periodic constant. It is shown that such systems are computationally universal, in the sense that for every recursive (r.e., co-r.e.) set $S \subseteq \mathbb{N}$ there exists a system with a unique (least, greatest) solution containing a component T with $S = \{n \mid 16n+13 \in T\}$. This implies undecidability of basic properties of these equations. All results also apply to language equations over a one-letter alphabet with concatenation and regular constants.

1. Introduction

Language equations are equations of the form

$$\varphi(X_1, \ldots, X_n) = \psi(X_1, \ldots, X_n), \qquad (*)$$

in which the unknowns X_i are formal languages, while the expressions φ, ψ use language-theoretic operations, such as concatenation, Kleene star and Boolean operations, and constant languages. It is well-known that systems of the *resolved form* $X_i = \varphi_i(X_1, \ldots, X_n)$ $(1 \leqslant i \leqslant n)$ with union, concatenation and singleton constants define the semantics of the context-free grammars [1]. If intersection is also allowed, such equations characterize an extension of the context-free grammars known as *conjunctive grammars* [9] and notable for efficient parsing algorithms.

The expressive power of language equations of the general form (*) was determined by Okhotin [10, 11], who proved that a language is representable by a unique (least, greatest) solution of a system with concatenation, Boolean operations and singleton constants if and only if this language is recursive (recursively enumerable, co-r.e., respectively). The same expressive power is attained using concatenation with constants and union [11]. It was subsequently discovered that language equations can be computationally universal even without any Boolean operations: Kunc [6] constructed a finite language $L \subseteq \{a, b\}^*$, for

(A. Jeż) Supported by MNiSW grant number N206 024 31/3826, 2006–2008 and N206 259035 2008–2010.
(A. Okhotin) Supported by the Academy of Finland under grant 118540.

which the greatest solution of a language equation $LX = XL$ is Π_1-hard (that is, hard for co-r.e. sets). This paper establishes a similar result in the seemingly trivial case of a one-letter alphabet.

Unary languages, defined over an alphabet $\{a\}$, form an important special class of formal languages. It is well-known that context-free grammars over this alphabet generate only regular languages. The first example of a unary language equation with a non-regular unique solution was constructed by Leiss [7]: this was an equation $X = \varphi(X)$ with φ containing concatenation, complementation and constant $\{a\}$. The question of whether conjunctive grammars (in other words, systems of language equations with union, intersection and concatenation) can generate any non-regular languages had been a long-standing open problem [9], until Jeż [2] constructed a conjunctive grammar generating $\{a^{4^n} \mid n \geqslant 0\}$. The ideas of this example were used by Jeż and Okhotin [3] to establish some general results on the expressive power of these equations, as well as the EXPTIME-completeness of their solutions [4]. For systems of the general form (*) using concatenation and union, it has recently been shown by the authors [5] that they are computationally complete.

As unary languages can be regarded as sets of natural numbers, unary language equations are naturally viewed as *equations over sets of numbers*. Concatenation of languages accordingly turns into *addition* of sets $S + T = \{m + n \mid m \in S,\ n \in T\}$, an operation that has been a subject of much study in number theory and combinatorics [14]. For instance, if P is the set of primes, then the equation $P + P + P = \{6, 7, \ldots\}$ expresses the Goldbach conjecture. Computational complexity of expressions and circuits over sets of numbers with addition and different sets of Boolean operations has been studied by Stockmeyer and Meyer [13] and McKenzie and Wagner [8]. Equations over sets of numbers are a more general formalism, and its expressive power was related to the allowed Boolean operations in the aforementioned work on unary language equations [2, 3, 4, 5].

This paper is concerned with equations over sets of numbers that use *only addition and no Boolean operations*. These are systems of equations of the form

$$X_{i_1} + \ldots + X_{i_k} + C = X_{j_1} + \ldots + X_{j_\ell} + D$$

in variables $(X_1, \ldots, X_n)$, where $C, D \subseteq \mathbb{N}$ are ultimately periodic constants. In terms of language equations over $\{a\}$, these are equations

$$X_{i_1} \ldots X_{i_k} K = X_{j_1} \ldots X_{j_\ell} L,$$

with regular constants $K, L \subseteq a^*$. This is the ultimately simplest case of language equations, and at the first glance it seems out of question that such equations could have any non-trivial unique (least, greatest) solutions. Probably for that reason no one has ever proclaimed their expressive power to be an open problem. However, as proved in this paper, these equations can have not only non-periodic unique solutions, but in fact are computationally universal. Furthermore, their main decision problems are as hard as similar problems for language equations over multiple-letter alphabets and using all Boolean operations [10, 11].

The new results are directly based on the authors' recent proof of the computational completeness of equations over sets of numbers with addition and union [5], though it is established using completely different techniques. The idea is to take an arbitrary system using addition and union and *encode* it in another system using addition only. The solutions of the two systems will not be identical, but there will be a bijection between solutions based upon an encoding of sets of numbers.

This encoding of sets, defined in Section 3, is an injection $\sigma : 2^{\mathbb{N}} \to 2^{\mathbb{N}}$, which represents every number n of the encoded set as the number $16n + 13$ in the encoding. The given encoding has two key properties. First of all, its form can be *checked* by an equation, which is satisfied exactly by those sets that are valid encodings; such an equation is constructed in Section 3. Second, the sum of any two valid encodings, encodes *both the sum and the union* of the encoded sets of numbers, and furthermore, adding a certain constant to such a sum of encodings produces a set that encodes only the sum of the original sets, while adding another constant allows representing only the union of the original sets. In overall, as shown in Section 4, the sum and the union of any two sets is represented by summing their encodings.

Finally, on the basis of this encoding, in Section 5 it is demonstrated how an arbitrary system of equations over sets of numbers with union and addition can be simulated using addition only. Each variable X_i of the original system will be represented in the new system by a variable X'_i, and the solutions of the new system will be of the form $X'_i = \sigma(S_i)$ for all variables X'_i, where $X_i = S_i$ is a solution of the original system.

All constants in the construction are ultimately periodic; some of them are finite and some are infinite. The last question is whether infinite constants are necessary to specify any non-periodic sets, and an affirmative answer is given in Section 6.

2. Equations over sets of numbers

Throughout this paper, the set of natural numbers $\mathbb{N} = \{0, 1, 2, \ldots\}$ is assumed to contain zero. A set of numbers $S \subseteq \mathbb{N}$ is *ultimately periodic* if there exist numbers $d \geqslant 0$ and $p \geqslant 1$, such that $n \in S$ if and only if $n + p \in S$ for every $n \geqslant d$. Otherwise, S is *non-periodic*. Note that S is ultimately periodic if and only if the corresponding language $L = \{a^n \mid n \in S\} \subseteq a^*$ is regular.

For every two subsets of natural numbers $S, T \subseteq \mathbb{N}$, their *sum* is the set $\{m + n \mid m \in S,\ n \in T\}$. Other typical operations on sets are the Boolean operations, such as union, intersection and complementation. Using complementation and addition, the first example of an equation with a non-periodic unique solution was constructed:

Example 2.1 (Leiss [7])**.** For every expression φ, denote $2\varphi = \varphi + \varphi$. Then the unique solution of the equation

$$X = 2\Big(\overline{2\big(\overline{2\overline{X}}\big)}\Big) + \{1\}$$

is $\{n \mid \exists i \geqslant 0 :\ 2^{3i} \leqslant n < 2^{3i+2}\} = \{n \mid \text{base-8 notation of } n \text{ starts with } \texttt{1}, \texttt{2} \text{ or } \texttt{3}\}$.

However, the expressive power of this family of equations is still quite limited [12], with some simple languages being non-representable.

The second example of non-periodic solutions of equations over sets of numbers was constructed by Jeż [2] as a conjunctive grammar [9] generating the language $\{a^{4^n} \mid n \geqslant 0\}$. In terms of equations it is stated as follows:

Example 2.2 (Jeż [2])**.** The least solution of the system

$$\begin{cases} X_1 &= \big((X_1{+}X_3) \cap (X_2{+}X_2)\big) \cup \{1\} \\ X_2 &= \big((X_1{+}X_1) \cap (X_2{+}X_6)\big) \cup \{2\} \\ X_3 &= \big((X_1{+}X_2) \cap (X_6{+}X_6)\big) \cup \{3\} \\ X_6 &= (X_1{+}X_2) \cap (X_3{+}X_3) \end{cases}$$

is $X_1 = \{4^n \mid n \geqslant 0\}$, $X_2 = \{2 \cdot 4^n \mid n \geqslant 0\}$, $X_3 = \{3 \cdot 4^n \mid n \geqslant 0\}$, $X_6 = \{6 \cdot 4^n \mid n \geqslant 0\}$.

The idea behind this example is to manipulate positional notations of numbers, and this idea was subsequently used to establish the following general result on the expressive power of such equations. The statement refers to the family of *linear conjunctive languages* [9], which properly contains the Boolean closure of linear context-free languages.

Proposition 2.3 (Jeż, Okhotin [3])**.** *For every $k \geqslant 2$ and for every linear conjunctive language $L \subseteq \{\mathtt{0}, \mathtt{1}, \ldots, k-1\}^+$ there exists a resolved system of equations*

$$\begin{cases} X_1 &= \varphi_1(X_1, \ldots, X_n) \\ &\vdots \\ X_n &= \varphi_n(X_1, \ldots, X_n) \end{cases}$$

with φ_i using singleton constants and the operations of union, intersection and addition, which has a least solution with $X_1 = \{n \mid$ the base-k notation of n is in $L\}$.

On the basis of this result, it was shown that systems of the general form with the same operations are computationally complete.

Theorem 2.4 (Jeż, Okhotin [5])**.** *For every recursive (r.e., co-r.e.) set $S \subseteq \mathbb{N}$ there exists an unresolved system*

$$\begin{cases} \varphi_1(X_1, \ldots, X_n) &= \psi_1(X_1, \ldots, X_n) \\ &\vdots \\ \varphi_m(X_1, \ldots, X_n) &= \psi_m(X_1, \ldots, X_n) \end{cases}$$

with φ_j, ψ_j using singleton constants and the operations of union and addition, which has a unique (least, greatest, respectively) solution with $X_1 = S$.

Exactly the same results hold for unresolved systems with intersection, sum and singleton constants [5], though they will not be used in this paper.

The goal is now to take any system of equations with union and addition, such as those constructed in Theorem 2.4, and to simulate it by another system using addition only. The solutions of the new system will *encode* the solutions of the original system as described in the next section.

3. Encoding of sets

An arbitrary set of numbers $\widehat{S} \subseteq \mathbb{N}$ will be represented by another set $S \subseteq \mathbb{N}$, which contains a number $16n+13$ if and only if n is in $\widehat{S}$. The membership of numbers i with $i \neq 13 \pmod{16}$ in S does not depend on $\widehat{S}$ and will be defined below. Since many constructions in the following will be done modulo 16, the following notation shall be adopted:

Definition 3.1. For each $i \in \{0, 1, \ldots, 15\}$,

$$\textsc{track}_i(S) = \{n \mid 16n + i \in S\},$$
$$\tau_i(S') = \{16n + i \mid n \in S'\}.$$

The subset $S \cap \{16n + i \mid n \geqslant 0\}$ is called the i^{th} *track of* S. A set S is said to have an *empty (full) track* i if $\textsc{track}_i(S) = \varnothing$ ($\textsc{track}_i(S) = \mathbb{N}$, respectively).

In these terms, it can be said that a set $\widehat{S}$ shall be encoded in the 13^{th} track of a set S. The rest of the tracks of S contain technical information needed for the below constructions to work: track 0 contains a singleton $\{0\}$, tracks 6, 8, 9 and 12 are full and the rest of the tracks are empty.

Definition 3.2. For every set $\widehat{S} \subseteq \mathbb{N}$, its *encoding* is the set

$$S = \sigma(\widehat{S}) = \{0\} \cup \tau_6(\mathbb{N}) \cup \tau_8(\mathbb{N}) \cup \tau_9(\mathbb{N}) \cup \tau_{12}(\mathbb{N}) \cup \tau_{13}(\widehat{S}).$$

The first property of the encoding announced in the introduction is that there exists an equation with the set of all valid encodings as its set of solutions. Such an equation will now be constructed.

Lemma 3.3. *A set $X \subseteq \mathbb{N}$ satisfies an equation*

$$X + \{0,4,11\} = \bigcup_{\substack{i \in \{0,4,6,8,9,\\ 10,12,13\}}} \tau_i(\mathbb{N}) \cup \bigcup_{i \in \{1,3,7\}} \tau_i(\mathbb{N}+1) \cup \{11\}$$

if and only if $X = \sigma(\widehat{X})$ for some $\widehat{X} \subseteq \mathbb{N}$.

Proof. $\Leftarrow$ Let X be any set that satisfies the equation. Then the sum $X + \{0,4,11\}$ has empty tracks 2, 5, 14 and 15:

$$\begin{aligned}\textsc{track}_2(X + \{0,4,11\}) = \textsc{track}_5(X + \{0,4,11\}) = \\ = \textsc{track}_{14}(X + \{0,4,11\}) = \textsc{track}_{15}(X + \{0,4,11\}) = \varnothing\end{aligned}$$

For this condition to hold, X must have many empty tracks as well. To be precise, each track t with t, $t+4$ or $t+11 \pmod{16}$ being in $\{2,5,14,15\}$ must be an empty track in X. Calculating such set of tracks, $\{2,5,14,15\} - \{0,4,11\} \pmod{16} = \{1,2,3,4,5,7,10,11,14,15\}$ are the numbers of tracks that must be empty in X.

Similar considerations apply to track 11, as $\textsc{track}_{11}(X + \{0,4,11\}) = \{0\}$. For every track t with $t = 11$, $t + 4 = 11$ or $t + 11 = 11 \pmod{16}$, it must hold that the t^{th} track of X is either an empty track or $\textsc{track}_t(X) = \{0\}$. The latter must hold for at least one such t. Let us calculate all such tracks t: these are tracks with numbers $\{11\} - \{0,4,11\} \pmod{16} = \{0,7,11\}$. Since tracks number 7 and 11 are already known to be empty, it follows that $\textsc{track}_0(X) = \{0\}$.

In order to prove that X is a valid encoding of some set, it remains to prove that tracks number $6,8,9,12$ in X are full. Consider first that $\textsc{track}_3(X + \{0,4,11\}) = \mathbb{N} + 1$. Let us calculate the track numbers t such that there is $t' \in \{0,4,11\}$ with $(t + t') \pmod{16} = 3$: these are $\{3\} - \{0,4,11\} \pmod{16} = \{3,8,15\}$. Since tracks $3, 15$ are known to be empty, then

$$\begin{aligned}\mathbb{N} + 1 = \textsc{track}_3(X + \{0,4,11\}) = \\ = \textsc{track}_3(X) \cup (\textsc{track}_{15}(X) + 1) \cup (\textsc{track}_8(X) + 1) = \\ = \varnothing \cup \varnothing \cup (\textsc{track}_8(X) + 1) = \textsc{track}_8(X) + 1,\end{aligned}$$

and thus track 8 of X is full. The analogous argument is used to prove that tracks $12, 9, 6$ are full. Consider $\textsc{track}_7(X + \{0,4,11\}) = \mathbb{N} + 1$. Then $\{7\} - \{0,4,11\} \pmod{16} = \{7,3,12\}$.

Since it is already known that tracks $3, 7$ are empty, the track 12 is full:

$$\begin{aligned}\mathbb{N}+1 = \textsc{track}_7(X+\{0,4,11\}) = \\ = \textsc{track}_7(X) \cup \textsc{track}_3(X) \cup (\textsc{track}_{12}(X)+1) = \\ = \varnothing \cup \varnothing \cup (\textsc{track}_{12}(X)+1) = \textsc{track}_{12}(X)+1.\end{aligned}$$

In the same way consider $\textsc{track}_9(X+\{0,4,11\}) = \mathbb{N}$. Then $\{9\} - \{0,4,11\} \pmod{16} = \{9,5,14\}$ and tracks $5, 14$ are empty, thus track 9 is full:

$$\begin{aligned}\mathbb{N} = \textsc{track}_9(X+\{0,4,11\}) = \\ = \textsc{track}_9(X) \cup \textsc{track}_5(X) \cup (\textsc{track}_{14}(X)+1) = \\ = \textsc{track}_9(X) \cup \varnothing \cup \varnothing = \textsc{track}_9(X).\end{aligned}$$

Now let us inspect $\textsc{track}_{10}(X+\{0,4,11\})$. Then $\{10\} - \{0,4,11\} \pmod{16} = \{10,6,15\}$. Since the tracks $10, 15$ are empty, then the 6^{th} track is full:

$$\begin{aligned}\mathbb{N} = \textsc{track}_{10}(X+\{0,4,11\}) = \\ = \textsc{track}_{10}(X) \cup \textsc{track}_6(X) \cup (1+\textsc{track}_{15}(X)) = \\ = \varnothing \cup \textsc{track}_6(X) \cup \varnothing = \textsc{track}_6(X).\end{aligned}$$

Thus it has been proved that $X = \sigma(\textsc{track}_{13}(X))$.

$\Leftarrow$ It remains to show the converse, that is, that if $X = \sigma(\widehat{X})$, then

$$X+\{0,4,11\} = \bigcup_{\substack{i\in\{0,4,6,8,9,\\ 10,12,13\}}} \tau_i(\mathbb{N}) \cup \bigcup_{i\in\{1,3,7\}} \tau_i(\mathbb{N}+1) \cup \{11\}.$$

Since $X = \bigcup_{i=0}^{15} \tau_i(\textsc{track}_i(X))$, then

$$\begin{aligned}X+\{0,4,11\} = \Big(\bigcup_i \tau_i(\textsc{track}_i(X))+0\Big) \cup \Big(\bigcup_i \tau_i(\textsc{track}_i(X))+4\Big) \cup \\ \cup \Big(\bigcup_i \tau_i(\textsc{track}_i(X))+11\Big),\end{aligned}$$

and Table 1 presents the form of each particular term in this union. Each i^{th} row represents track number i in X, and each column labelled $+j$ for $j \in \{0,4,11\}$ corresponds to the addition of a number j. The cell (i,j) gives the set $\textsc{track}_i(X)+j$ and the number of the track in which this set appears in the result (this is track $i+j \pmod{16}$). Then each ℓ^{th} track of $X+\{0,4,11\}$ is obtained as a union of all the appropriate sets in the Table 1.

According to the table, the values of the set $\widehat{X}$ are reflected in three tracks of the sum $X+\{0,4,11\}$: in tracks 13, 1 and 8 (in the last two cases, with offset 1). However, at the same time the sum contains full tracks 8 and 13, as well as $\mathbb{N}+1$ in track 1, and the contributions of $\widehat{X}$ to the sum are subsumed by these numbers, as $\tau_{13}(\widehat{X}) \subseteq \tau_{13}(\mathbb{N})$, $\tau_1(\widehat{X}+1) \subseteq \tau_1(\mathbb{N}+1)$ and $\tau_8(\widehat{X}+1) \subseteq \tau_8(\mathbb{N})$. Therefore, the value of the expression does not depend on $\widehat{X}$. Taking the union of all entries of the Table 1 proves that $X+\{0,4,11\}$ equals

$$\bigcup_{\substack{i\in\{0,4,6,8,9,\\ 10,12,13\}}} \tau_i(\mathbb{N}) \cup \bigcup_{i\in\{1,3,7\}} \tau_i(\mathbb{N}+1) \cup \{11\},$$

	$+0$	$+4$	$+11$
$0\colon \{0\}$	$0\colon \{0\}$	$4\colon \{0\}$	$11\colon \{0\}$
$6\colon \mathbb{N}$	$6\colon \mathbb{N}$	$10\colon \mathbb{N}$	$1\colon \mathbb{N}+1$
$8\colon \mathbb{N}$	$8\colon \mathbb{N}$	$12\colon \mathbb{N}$	$3\colon \mathbb{N}+1$
$9\colon \mathbb{N}$	$9\colon \mathbb{N}$	$13\colon \mathbb{N}$	$4\colon \mathbb{N}+1$
$12\colon \mathbb{N}$	$12\colon \mathbb{N}$	$0\colon \mathbb{N}+1$	$7\colon \mathbb{N}+1$
$13\colon \widehat{X}$	$13\colon \widehat{X}$	$1\colon \widehat{X}+1$	$8\colon \widehat{X}+1$

Table 1: Tracks in the sum $\sigma(\widehat{X})+\{0,4,11\}$, only non-empty tracks of $\sigma(\widehat{X})$ are included.

as stated in the lemma. ∎

4. Simulating operations

The goal of this section is to establish the second property of the encoding σ, that is, that a sum of encodings of two sets and a fixed constant set effectively encodes the union of these two sets, while the addition of a different fixed constant set allows encoding the sum of the two original sets. This property is formally stated in the following lemma, along with the actual constant sets:

Lemma 4.1. *For all sets $X, Y, Z \subseteq \mathbb{N}$,*

$$\sigma(Y)+\sigma(Z)+\{0,1\} = \sigma(X)+\sigma(\{0\})+\{0,1\} \text{ if and only if } Y+Z=X$$

and

$$\sigma(Y)+\sigma(Z)+\{0,2\} = \sigma(X)+\sigma(X)+\{0,2\} \text{ if and only if } Y\cup Z=X.$$

Proof. The goal is to show that for all $Y, Z \subseteq \mathbb{N}$, the sum

$$\sigma(Y)+\sigma(Z)+\{0,1\}$$

encodes the set $Y+Z+1$ on one of its tracks, while the contents of all other tracks do not depend on Y or on Z. Similarly, the sum

$$\sigma(Y)+\sigma(Z)+\{0,2\}$$

has a track that encodes $Y\cup Z$, while the rest of its tracks also do not depend on Y and Z.

The common part of both of the above sums is $\sigma(Y)+\sigma(Z)$, so let us calculate it first. Since

$$\sigma(Y) = \{0\}\cup\tau_6(\mathbb{N})\cup\tau_8(\mathbb{N})\cup\tau_9(\mathbb{N})\cup\tau_{12}(\mathbb{N})\cup\tau_{13}(Y) \quad \text{and}$$
$$\sigma(Z) = \{0\}\cup\tau_6(\mathbb{N})\cup\tau_8(\mathbb{N})\cup\tau_9(\mathbb{N})\cup\tau_{12}(\mathbb{N})\cup\tau_{13}(Z),$$

by the distributivity of union, the sum $\sigma(Y)+\sigma(Z)$ is a union of 36 terms, each being a sum of two individual tracks. Every such sum is contained in a single track as well, and Table 2 gives a case inspection of the form of all these terms. Each of its six rows corresponds to one of the nonempty tracks of $\sigma(Y)$, while its six columns refer to the nonempty tracks in $\sigma(Z)$. Then the cell gives the sum of these tracks, in the form of the track number and track contents: that is, for row representing $\textsc{track}_i(\sigma(Y))$ and for column representing $\textsc{track}_j(\sigma(Z))$, the cell (i,j) represents the set $\textsc{track}_i(\sigma(Y))+\textsc{track}_j(\sigma(Z))$, which is

	0: $\{0\}$	6: $\mathbb{N}$	8: $\mathbb{N}$	9: $\mathbb{N}$	12: $\mathbb{N}$	13: Z
0: $\{0\}$	0: $\{0\}$	6: $\mathbb{N}$	8: $\mathbb{N}$	9: $\mathbb{N}$	12: $\mathbb{N}$	13: Z
6: $\mathbb{N}$	6: $\mathbb{N}$	12: $\mathbb{N}$	14: $\mathbb{N}$	15: $\mathbb{N}$	2: $\mathbb{N}+1$	3: ?
8: $\mathbb{N}$	8: $\mathbb{N}$	14: $\mathbb{N}$	0: $\mathbb{N}+1$	1: $\mathbb{N}+1$	4: $\mathbb{N}+1$	5: ?
9: $\mathbb{N}$	9: $\mathbb{N}$	15: $\mathbb{N}$	1: $\mathbb{N}+1$	2: $\mathbb{N}+1$	5: $\mathbb{N}+1$	6: ?
12: $\mathbb{N}$	12: $\mathbb{N}$	2: $\mathbb{N}+1$	4: $\mathbb{N}+1$	5: $\mathbb{N}+1$	8: $\mathbb{N}+1$	9: ?
13: Y	13: Y	3: ?	5: ?	6: ?	9: ?	10: $(Y+Z)+1$

Table 2: Tracks in the sum $\sigma(Y) + \sigma(Z)$. Question marks denote subsets of $\mathbb{N} + 1$ that depend on Y or Z and whose actual values are unimportant.

	$\sigma(Y)$	$\sigma(Z)$	$\sigma(Y)+\sigma(Z)$	$\sigma(Y)+\sigma(Z)+\{0,1\}$	$\sigma(Y)+\sigma(Z)+\{0,2\}$
0	$\{0\}$	$\{0\}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$
1	$\varnothing$	$\varnothing$	$\mathbb{N}+1$	$\mathbb{N}$	$\mathbb{N}+1$
2	$\varnothing$	$\varnothing$	$\mathbb{N}+1$	$\mathbb{N}+1$	$\mathbb{N}$
3	$\varnothing$	$\varnothing$	?	$\mathbb{N}+1$	$\mathbb{N}+1$
4	$\varnothing$	$\varnothing$	$\mathbb{N}+1$	$\mathbb{N}+1$	$\mathbb{N}+1$
5	$\varnothing$	$\varnothing$	$\mathbb{N}+1$	$\mathbb{N}+1$	$\mathbb{N}+1$
6	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$
7	$\varnothing$	$\varnothing$	$\varnothing$	$\mathbb{N}$	$\mathbb{N}+1$
8	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$
9	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$
10	$\varnothing$	$\varnothing$	$Y+Z+1$	$\mathbb{N}$	$\mathbb{N}$
11	$\varnothing$	$\varnothing$	$\varnothing$	$Y+Z+1$	$\mathbb{N}$
12	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$
13	Y	Z	$Y \cup Z$	$\mathbb{N}$	$Y \cup Z$
14	$\varnothing$	$\varnothing$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$
15	$\varnothing$	$\varnothing$	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{N}$

Table 3: Tracks in the sums of $\sigma(Y) + \sigma(Z)$ with constants.

bound to be on track $i + j \pmod{16}$. For example, the sum of track 8 of $\sigma(Y)$ and track 9 of $\sigma(Z)$ falls onto track $1 = 8 + 9 \pmod{16}$ and equals

$$\tau_8(\mathbb{N}) + \tau_9(\mathbb{N}) = \{8 + 9 + 16(m + n) \mid m, n \geqslant 0\} = \{1 + 16n \mid n \geqslant 1\} = \tau_1(\mathbb{N} + 1),$$

while adding track 13 of $\sigma(Y)$ to track 13 of $\sigma(Z)$ results in

$$\tau_{13}(Y) + \tau_{13}(Z) = \{26 + 16(m + n) \mid m \in Y,\ n \in Z\} = \tau_{10}(Y + Z + 1),$$

which is reflected in the table. Each question mark denotes a track with unspecified contents. Though these contents can be calculated, it is actually irrelevant, because it does not influence the value of the subsequent sums $\sigma(Y) + \sigma(Z) + \{0, 1\}$ and $\sigma(Y) + \sigma(Z) + \{0, 2\}$. What is important is that none of these tracks contain 0.

Now the value of each i^{th} track of $\sigma(Y) + \sigma(Z)$ is obtained as the union of all sums in Table 2 that belong to the i^{th} track. The final values of these tracks are presented in the corresponding column of Table 3.

Now the contents of the tracks in $\sigma(Y) + \sigma(Z) + \{0,1\}$ can be completely described. The calculations are given in Table 3, and the result is that for all Y and Z,

$$\begin{aligned}
&\textsc{track}_{11}(\sigma(Y) + \sigma(Z) + \{0,1\}) = Y + Z + 1,\\
&\textsc{track}_i(\sigma(Y) + \sigma(Z) + \{0,1\}) = \mathbb{N} + 1 && \text{for } i \in \{2,3,4,5\},\\
&\textsc{track}_i(\sigma(Y) + \sigma(Z) + \{0,1\}) = \mathbb{N} && \text{for all other } i.
\end{aligned}$$

It easily follows that

$$X = Y + Z$$

if and only if

$$\sigma(X) + \sigma(\{0\}) + \{0,1\} = \sigma(Y) + \sigma(Z) + \{0,1\},$$

as, clearly, $X = X + \{0\}$.

For the set $\sigma(Y) + \sigma(Z) + \{0,2\}$, in the same way, for all Y and Z,

$$\begin{aligned}
&\textsc{track}_{13}(\sigma(Y) + \sigma(Z) + \{0,2\}) = Y \cup Z,\\
&\textsc{track}_j(\sigma(Y) + \sigma(Z) + \{0,2\}) = \mathbb{N} + 1 && \text{for } j \in \{1,3,4,5,7\},\\
&\textsc{track}_j(\sigma(Y) + \sigma(Z) + \{0,2\}) = \mathbb{N} && \text{for all other } j,
\end{aligned}$$

and therefore for all X, Y, Z,

$$X = Y \cup Z$$

if and only if

$$\sigma(X) + \sigma(X) + \{0,2\} = \sigma(Y) + \sigma(Z) + \{0,2\},$$

since $X = X \cup X$.

Both claims of the lemma follow. ∎

5. Simulating a system of equations

Using the encoding defined above, it is now possible to represent a system with union and addition by a system with addition only. Since Lemma 4.1 on the simulation of individual operations is applicable only to equations of a simple form, the first task is to convert a given system to such a form:

Lemma 5.1. *For every system of equations over sets of numbers in variables $(X_1, \ldots, X_n)$ using union, addition and constants from a set $\mathcal{C}$ there exists a system in variables $(X_1, \ldots, X_n, X_{n+1}, \ldots, X_{n+m})$ with all equations of the form $X_i = X_j + X_k$, $X_i = X_j \cup X_k$ or $X_i = C$ with $C \in \mathcal{C}$, such that the set of solutions of this system is*

$$\big\{(S_1, \ldots, S_n, \ldots, f_i(S_1, \ldots, S_n), \ldots) \;\big|\; (S_1, \ldots, S_n) \text{ is a solution of the original system}\big\},$$

for some monotone functions $f_1, \ldots, f_m$.

The construction is by a straightforward decomposition of equations, with new variables representing subexpressions of the sides of the original equations. Once the equations are thus transformed, the system can be encoded as follows.

Lemma 5.2. *For every system of equations over sets of numbers in variables $(\dots, X, \dots)$ and with all equations of the form $X = Y + Z$, $X = Y \cup Z$ or $X = C$, there exists a system in variables $(\dots, X', \dots)$, using only addition and constants $\{0,1\}$, $\{0,2\}$, $\{0,4,11\}$, $\sigma(\{0\})$, $\sigma(C)$ with C used in the original system and the ultimately periodic constant from Lemma 3.3, such that $(\dots, S'_X, \dots)$ is a solution of the latter system if and only if $S'_X = \sigma(S_X)$ for each variable X, for some solution $(\dots, S_X, \dots)$ of the former system.*

Proof. The proof is by a direct transformation of this system according to Lemmata 3.3 and 4.1. First, the new system contains the following equation for each variable X':

$$X' + \{0,4,11\} = \bigcup_{\substack{i \in \{0,4,6,8,9,\\ 10,12,13\}}} \tau_i(\mathbb{N}) \cup \bigcup_{i \in \{1,3,7\}} \tau_i(\mathbb{N}+1) \cup \{11\}. \tag{5.1}$$

Next, for each equation $X = Y + Z$ in the original system, there is a corresponding equation

$$X' + \sigma(\{0\}) + \{0,1\} = Y' + Z' + \{0,1\} \tag{5.2}$$

in the new system. Similarly, for each equation of the form $X = Y \cup Z$, the new system contains an equation

$$X' + X' + \{0,2\} = Y' + Z' + \{0,2\}. \tag{5.3}$$

Finally, every equation $X = C$ in the original system is represented in the new system by the following equation:

$$X' = \sigma(C). \tag{5.4}$$

By Lemma 3.3, (5.1) ensures that each solution $(\dots, S'_X, \dots)$ of the constructed system satisfies $S'_X = \sigma(S_X)$ for some sets S_X. It is claimed that $(\dots, S_X, \dots)$ satisfies each equation of the original system if and only if $(\dots, \sigma(S_X), \dots)$ satisfies the corresponding equation (5.2–5.4) of the constructed system. Consider each pair of corresponding equations:

- Consider an equation $X = Y \cup Z$ from the original system. Then there is a corresponding equation (5.2), and, by Lemma 4.1, $(\dots, S_X, \dots)$ satisfies the original equation if and only if $(\dots, \sigma(S_X), \dots)$ satisfies (5.2).
- Similarly, by Lemma 4.1, an equation of the form $X = Y + Z$ is satisfied by $(\dots, S_X, \dots)$ if and only if $(\dots, \sigma(S_X), \dots)$ satisfies the corresponding equation (5.3).
- For each equation of the form $X = C$ it is claimed that a set S_X satisfies it if and only if $\sigma(S_X)$ satisfies the corresponding equation (5.4). Indeed, $\sigma(S_X) = \sigma(C)$ if and only if $\textsc{track}_{13}(\sigma(S_X)) = \textsc{track}_{13}(\sigma(C))$, and since $\textsc{track}_{13}(\sigma(S_X)) = S_X$ and $\textsc{track}_{13}(\sigma(C)) = C$, this is equivalent to $S_X = C$.

This shows that $(\dots, S_X, \dots)$ satisfies the original system if and only if $(\dots, \sigma(S_X), \dots)$ satisfies the constructed system, which proves the correctness of the construction. ∎

Note that σ is a bijection between the sets of solutions of the two systems. Then, in particular, if the original system has a unique solution, then the constructed system has a unique solution as well, which encodes the solution of the original system.

Furthermore, it is important that the encoding σ respects inclusion, that is, if $X \subseteq Y$, then $\sigma(X) \subseteq \sigma(Y)$. Consider the partial order on solutions of a system, defined as $(S_1, \dots, S_n) \preccurlyeq (S'_1, \dots, S'_n)$ if $S_i \subseteq S'_i$ for all i. Now if one solution of the original system is less than another, then the corresponding solutions of the constructed system maintain this relation. Therefore, if the original system has a least (greatest) solution with respect to this partial order, then so does the new one, and its least (greatest) solution is the image of the least (greatest) solution of the original system.

These observations allow applying Lemmata 5.1 and 5.2 to encode each system in Theorem 2.4 within a system using addition only.

Theorem 5.3. *For every recursive (r.e., co-r.e.) set $S \subseteq \mathbb{N}$ there exists a system of equations*

$$\begin{cases} \varphi_1(X_1, \ldots, X_n) &= \psi_1(X_1, \ldots, X_n) \\ & \vdots \\ \varphi_m(X_1, \ldots, X_n) &= \psi_m(X_1, \ldots, X_n) \end{cases}$$

with φ_j, ψ_j using the operation of addition and ultimately periodic constants, which has a unique (least, greatest, respectively) solution with $X_1 = T$, where $S = \{n \mid 16n + 13 \in T\}$.

Note that S is computationally reducible to T via the "$16n + 13$" transduction, hence the following statements:

Corollary 5.4. *For every recursive set $S \subseteq \mathbb{N}$ there exists a system of equations over sets of natural numbers using addition and ultimately periodic constants that has a unique solution, which is computationally as hard as S.*

Corollary 5.5. *There exists a system of equations over sets of natural numbers using addition and ultimately periodic constants which has a least (greatest) solution with its first component being r.e.-complete (co-r.e.-complete, respectively).*

Finally, the decision problems for these systems of equations turn out to be as hard as in the case of union and addition:

Theorem 5.6. *The problem of testing whether a system of equations over sets of natural numbers using addition and ultimately periodic constants has a solution is Π_1-complete. The problem of whether it has a unique, least or greatest solution is Π_2-complete. The problem of whether it has finitely many solutions is Σ_3-complete.*

The above results equally apply to *language equations* over a one-letter alphabet with concatenation as the only allowed operation and with regular constants.

6. Systems with finite constants

The constructions above essentially use three infinite ultimately periodic constants: one of them is the right-hand side of the equation from Lemma 3.3, and the other two are the sets $\sigma(\{0\})$ and $\sigma(\{1\})$ used in Lemma 5.2 to represent constants $\{0\}$ and $\{1\}$. It will now be shown that the use of such constants is necessary, and systems using only addition and *finite* constants cannot specify any non-trivial infinite sets.

This is done by demonstrating that every solution $(\ldots, S, \ldots)$ of such a system can be *pruned* in the sense that each of its infinite components can be replaced by an empty set and the resulting vector remains a solution.

Lemma 6.1. *If a system of equations in variables $(\ldots, X_j, \ldots, Y_i, \ldots)$ using addition and only finite constants has a solution $(\ldots, F_j, \ldots, S_i, \ldots)$, where each F_j is finite and each S_i infinite, then $(\ldots, F_j, \ldots, \varnothing, \ldots)$ is a solution of this system.*

In a similar way, infinite components of a solution can be augmented to co-finite sets. For every nonempty set $S \subseteq \mathbb{N}$, consider its *upward closure* $S + \mathbb{N}$, which is always co-finite.

Lemma 6.2. *If a system of equations in variables $(\ldots, X_j, \ldots, Y_i, \ldots)$ using addition and only finite constants has a solution $(\ldots, F_j, \ldots, S_i, \ldots)$, where each F_j is finite and each S_i infinite, then $(\ldots, F_j, \ldots, S_i + \mathbb{N}, \ldots)$ is a solution as well.*

Theorem 6.3. *If a system of equations using addition and finite constants has a least (greatest, unique) solution $(\ldots, S_i, \ldots)$, then each S_i is finite (finite or co-finite, finite, respectively).*

Since equations with finite constants have so trivial solutions, it is natural to expect their decision problems to be much easier than in Theorem 5.6. Establishing the exact complexity of these problems is left for future work.

7. Conclusion

The study of language equations has progressed by showing the computational universality of simpler and simpler models [10, 6, 5]. The equations proved universal in this paper are the simplest considered so far: the constructions use systems of equations $X = YZ$ and $X = C$ over an alphabet $\Sigma = \{a\}$, with ultimately periodic constants $C \subseteq a^*$. Little room is left for further improvement, as infinite constants were proved to be essential.

The results have been obtained in terms of equations over sets of numbers using the operation of addition, which is the main subject of additive combinatorics [14]. Hopefully, this work will lead to some further connections between computability and number theory.

References

[1] S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.
[2] A. Jeż, "Conjunctive grammars can generate non-regular unary languages", *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.
[3] A. Jeż, A. Okhotin, "Conjunctive grammars over a unary alphabet: undecidability and unbounded growth", *Theory of Computing Systems*, to appear.
[4] A. Jeż, A. Okhotin, "Complexity of solutions of equations over sets of natural numbers", *STACS 2008*.
[5] A. Jeż, A. Okhotin, "On the computational completeness of equations over sets of natural numbers", *ICALP 2008*, part II, LNCS 5126, 63–74.
[6] M. Kunc, "The power of commuting with finite sets of words", *Theory of Computing Systems*, 40:4 (2007), 521–551.
[7] E. L. Leiss, "Unrestricted complementation in language equations over a one-letter alphabet", *Theoretical Computer Science*, 132 (1994), 71–93.
[8] P. McKenzie, K. Wagner, "The complexity of membership problems for circuits over sets of natural numbers", *Computational Complexity*, 16:3 (2007), 211–244.
[9] A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.
[10] A. Okhotin, "Decision problems for language equations with Boolean operations", *ICALP 2003*.
[11] A. Okhotin, "Unresolved systems of language equations: expressive power and decision problems", *Theoretical Computer Science*, 349:3 (2005), 283–308.
[12] A. Okhotin, O. Yakimova, "On language equations with complementation", *DLT 2006*, 420–432.
[13] L. J. Stockmeyer, A. R. Meyer, "Word problems requiring exponential time", *STOC 1973*, 1–9.
[14] T. Tao, V. Vu, *Additive Combinatorics*, Cambridge University Press, 2006.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 589–600
www.stacs-conf.org

DECIDING UNAMBIGUITY AND SEQUENTIALITY OF POLYNOMIALLY AMBIGUOUS MIN-PLUS AUTOMATA

DANIEL KIRSTEN [1] AND SYLVAIN LOMBARDY [2]

[1] University Leipzig, Inst. for Computer Science, Postfach 10 09 20, 04009 Leipzig, Germany

[2] IGM-LabInfo, Université Paris-Est Marne-la-Vallée, 77454 Marne-la-Vallée Cedex 2, France

Abstract. This paper solves the unambiguity and the sequentiality problem for polynomially ambiguous min-plus automata. This result is proved through a decidable algebraic characterization involving so-called metatransitions and an application of results from the structure theory of finite semigroups. It is noteworthy that the equivalence problem is known to be undecidable for polynomially ambiguous automata.

1. Introduction

Min-plus and max-plus automata are studied under various names in the literature, e.g. distance, finance, or cost automata. They have also appeared in various contexts: logical problems in formal language theory (star height, finite power property, star problem for traces) [6, 12, 13, 23, 20], study of dynamics of some discrete event systems (DES) [1, 2], automatic speech recognition [21], and database theory [3].

The sequentiality/unambiguity problem is one of the most intriguing open problems for min-plus automata: decide (constructively) whether some given min-plus automaton admits a sequential/unambiguous equivalent. This problem is wide open despite the fact it was studied by several researchers, e.g. [15, 19, 21].

In 2004, Klimann, Lombardy, Mairesse, and Prieur showed that this problem is decidable for finitely ambiguous min-plus automata [15]. For the sequentiality problem, Mohri presented an imperfect algorithm (which is not a decision algorithm) in 1997 [21].

In the present paper, we show a new partial solution to the sequentiality/unambiguity problem: we show that this problem is decidable provided that the input automaton is polynomially ambiguous. Polynomially ambiguous min-plus automata are much more involved objects than finitely ambiguous ones, e.g. the equivalence problem is undecidable for polynomially, but decidable for finitely ambiguous min-plus automata [16, 8]. In fact, all the key ideas in [15] for finitely ambiguous min-plus automata (namely the decomposition technique and the pumping arguments) do not carry over to polynomially ambiguous min-plus automata and we have to develop advanced proof techniques. We develop a theory of

Key words and phrases: min-plus automata, determinization, finite semigroups.

The main results of this paper were achieved during a three month stay of Daniel Kirsten at the Institute Gaspard Monge at the Université Paris-Est Marne-la-Vallée which was funded by the CNRS.

so-called metatransitions and establish a decidable algebraic characterization of the polynomially ambiguous min-plus automata which admit an unambiguous equivalent. To prove the characterization, we utilize some techniques from the limitedness problem for distance and desert automata [18, 24, 12, 13], results from the structure theory of finite semigroups as the factorization forest theorem along with various new ideas. The proof for the sufficiency of the construction leads to an intriguing combination of two BURNSIDE problems.

2. Preliminaries

2.1. Notations

Let Σ be a finite alphabet. The notion of a $\sharp$-expression is due to [7]. Every $a \in \Sigma$ is a *$\sharp$-expression*. For $\sharp$-expressions r and s, the expressions rs and $r^\sharp$ are *$\sharp$-expressions*. For a $\sharp$-expression r and $k \geq 0$, let $r(k)$ be the word obtained by replacing every $\sharp$ by k.

Let $\mathbb{N} = \{0, 1, \dots\}$. Let $\mathbb{Z}_\omega = (\mathbb{Z} \cup \{\omega, \infty\}, \mathsf{min}, +, \infty, 0)$ be the semiring whereas min is the minimum for the ordering $\cdots \leq -1 \leq 0 \leq 1 \cdots \leq \omega \leq \infty$ and $m + n$ is defined as usual if $m, n \in \mathbb{Z}$ but as maximum of m and n if $m \in \{\omega, \infty\}$ or $n \in \{\omega, \infty\}$. The *tropical semiring* $\mathbb{Z}_\infty$ is the restriction of $\mathbb{Z}_\omega$ to $\mathbb{Z} \cup \{\infty\}$.

Let Q be a finite set. For $k \geq 1$, matrices $M_1, \dots, M_k, M \in \mathbb{Z}_\omega^{Q \times Q}$, and $p_0, \dots, p_k \in Q$, we denote $M_1[p_0, p_1] + \cdots + M_k[p_{k-1}, p_k]$ by $(M_1, \dots, M_k)[p_0, \dots, p_k]$, and we denote $M[p_0, p_1] + \cdots + M[p_{k-1}, p_k]$ by $M[p_0, \dots, p_k]$.

Let $M \in \mathbb{Z}_\omega^{Q \times Q}$. We set $\mathsf{mind}(M) = \mathsf{min}\{M[p, p] \mid p \in Q\}$. If some entry of M belongs to $\mathbb{Z}$, then $\mathsf{min}(M)$ (resp. $\mathsf{max}(M)$) is the minimum (resp. maximum) of the set $\{M[p, q] \mid p, q \in Q, M[p, q] \in \mathbb{Z}\}$, and $\mathsf{span}(M) = \mathsf{max}(M) - \mathsf{min}(M)$. Otherwise, $\mathsf{span}(M) = 0$.

The boolean semiring is $\mathbb{B} = (\{0, 1\}, +, \cdot, 0, 1)$, and we denote by $\alpha : \mathbb{Z}_\omega \to \mathbb{B}$ the morphism defined by $\alpha(\infty) = 0$ and $\alpha(z) = 1$ for $z \neq \infty$.

Given $P \subseteq Q$ and $M \in \mathbb{B}^{Q \times Q}$, we let $P \cdot M = \{q \in Q \mid$ there is some $p \in P$ such that $M[p, q] = 1\}$ and $M \cdot P = \{q \in Q \mid$ there is some $p \in P$ such that $M[q, p] = 1\}$.

We generalize all these notions (except mind) to matrices which are not quadratic.

Let T be a set and $\cdot : T \times T \dashrightarrow T$ be partial mapping. We assume that $\cdot$ is associative, i.e., if for $p, q, r \in T$, either both products $(pq)r$ and $p(qr)$ are undefined or both products are defined and $(pq)r = p(qr)$. Let $T_0 = T \uplus \{0\}$. We extend $\cdot$ to T_0 by setting $pq = 0$ for $p, q \in T$ for which pq is undefined in T. Clearly, T_0 is a semigroup with zero 0.

2.2. Min-Plus Automata

A *min-plus automaton* is a tuple $\mathcal{A} = [Q, \mu, \lambda, \varrho]$ whereas Q is a nonempty, finite set of *states*, $\mu : \Sigma^* \to \mathbb{Z}_\infty^{Q \times Q}$ is a homomorphism, and $\lambda, \varrho \in \mathbb{Z}_\infty^Q$. A min-plus automaton $\mathcal{A}$ computes a mapping $|\mathcal{A}| : \Sigma^* \to \mathbb{Z}_\infty$ by $|\mathcal{A}|(w) = \lambda\mu(w)\varrho$ for $w \in \Sigma^*$.

Two min-plus automata are *equivalent* if and only if they compute the same mapping. We call a state $q \in Q$ *accessible* (resp. *co-accessible*) if there is a $v \in \Sigma^*$ such that $(\lambda\mu(v))[q] \in \mathbb{Z}$ (resp. $(\mu(v)\varrho)[q] \in \mathbb{Z}$). If every state is accessible and co-accessible, then we call $\mathcal{A}$ *trim*.

Let $I = \{\, q \in Q \mid \lambda[q] \in \mathbb{Z} \,\}$ and $F = \{\, q \in Q \mid \varrho[q] \in \mathbb{Z} \,\}$. If $|I| = 1$, and for every $a \in \Sigma$, $p \in Q$, there exists at most one $q \in Q$ satisfying $\mu(a)[p, q] \in \mathbb{Z}$, then we call $\mathcal{A}$ *sequential*.

Let $w = a_1 \cdots a_{|w|} \in \Sigma^*$. A sequence $p_0, \dots, p_{|w|}$ is a *path (in $\mathcal{A}$) from p_0 to $p_{|w|}$ for w* if $(\mu(a_1), \dots, \mu(a_{|w|}))[p_0, \dots, p_{|w|}] \in \mathbb{Z}$. We call $p_0, \dots, p_{|w|}$ *accepting* if $p_0 \in I$, $p_{|w|} \in F$.

If there exists some polynomial $P : \mathbb{N} \to \mathbb{N}$ such that for every $w \in \Sigma^*$, there are at most $P(|w|)$ accepting paths for w, then $\mathcal{A}$ is called *polynomially ambiguous*. If the same condition is satisfied for a constant $n \in \mathbb{N}$, then $\mathcal{A}$ is called *finitely ambiguous*. If there is at most one path for each word, then $\mathcal{A}$ is called *unambiguous*. The mapping $f : \{a,b\}^* \to \mathbb{Z}_\infty$ defined as $f(w) = \mathsf{min}\{k \mid ba^kb \text{ is a factor of } w\}$ can be computed by a polynomially ambiguous min-plus automaton, but not by a finitely ambiguous min-plus automaton [14].

The following characterization is used implicitly in [10, 11, 22] (cf. Proof of Theorem 3.1 in [11] or Lemma 4.3 in [10]).

Theorem 2.1. *A trim min-plus automaton $\mathcal{A}$ is polynomially ambiguous if and only if for every state q and every $w \in \Sigma^*$, there is at most one path for w from q to q.*

We need the following characterization.

Lemma 2.2. *Let $\mathcal{A} = [Q, \mu, \lambda, \varrho]$ be a trim, unambiguous min-plus automaton. Let $w \in \Sigma^*$, $k \geq 1$, and $q_0, \ldots, q_k \in Q$ such that there is path for w from q_{i-1} to q_i for every $1 \leq i \leq k$.*

There are $\pi_1, \pi_2, \pi_3 \in Q^*$ such that $|\pi_1\pi_3| \leq |Q|$, $|\pi_2| \leq |Q|$, and $q_0 \ldots q_k \in \pi_1\pi_2^*\pi_3$.

3. Overview

3.1. Metatransitions

The combination of a forward and backward parsing was one of the key ideas by Hashiguchi in various papers on the finite power property and distance automata (e.g. in [4, 5]). Metatransitions formalize this idea in an algebraic fashion. Metatransitions form a semigroup, and the homomorphism $\alpha : \mathbb{Z}_\omega \to \mathbb{B}$ extends in a natural way to a homomorphism between the semigroups of metatransitions. Henceforth, we can utilize semigroup theoretic approaches by Simon, Leung, and Kirsten (e.g. [18, 23, 24, 12, 13]) on metatransitions. Consequently, the concept of a metatransition compromises the combinatorial approach by Hashiguchi and the algebraic approach by Simon and Leung in the research on min-plus automata. Several results in this section were already shown in [9].

Let Q be a finite set. A *metatransition over $\mathbb{Z}_\omega$ and Q* is a tuple $\left(\begin{smallmatrix} P_0 \\ R_0 \end{smallmatrix} M \begin{smallmatrix} P_1 \\ R_1 \end{smallmatrix}\right)$, whereas

MT1.: $P_0, P_1, R_0, R_1 \subseteq Q$,
MT2.: $M \in \mathbb{Z}_\omega^{(P_0 \cap R_0) \times (P_1 \cap R_1)}$,
MT3.: $(P_0 \cap R_0) \cdot \alpha(M) = (P_1 \cap R_1)$ and $(P_0 \cap R_0) = \alpha(M) \cdot (P_1 \cap R_1)$.

Two metatransitions $\left(\begin{smallmatrix} P_0 \\ R_0 \end{smallmatrix} M \begin{smallmatrix} P_1 \\ R_1 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} P_0' \\ R_0' \end{smallmatrix} M' \begin{smallmatrix} P_1' \\ R_1' \end{smallmatrix}\right)$ are called *concatenable* if and only if $P_1 = P_0'$ and $R_1 = R_0'$. In this case, their product yields $\left(\begin{smallmatrix} P_0 \\ R_0 \end{smallmatrix} MM' \begin{smallmatrix} P_1' \\ R_1' \end{smallmatrix}\right)$.

Let $\mathsf{MT}(\mathbb{Z}_\omega, Q)$ be the set consisting of all metatransitions over Q. Then, $\mathsf{MT}(\mathbb{Z}_\omega, Q)_0$ is a semigroup with a zero.

We define metatransitions over $\mathbb{B}$ and Q in the same way.[1] We extend the homomorphism $\alpha : \mathbb{Z}_\omega \to \mathbb{B}$ to $\alpha : \mathsf{MT}(\mathbb{Z}_\omega, Q)_0 \to \mathsf{MT}(\mathbb{B}, Q)_0$ by setting

$$\alpha\left(\left(\begin{smallmatrix} P_0 \\ R_0 \end{smallmatrix} M_1 \begin{smallmatrix} P_1 \\ R_1 \end{smallmatrix}\right)\right) = \left(\begin{smallmatrix} P_0 \\ R_0 \end{smallmatrix} \alpha(M_1) \begin{smallmatrix} P_1 \\ R_1 \end{smallmatrix}\right) \quad \text{and} \quad \alpha(0) = 0.$$

[1] In (MT3), $\alpha(M)$ is replaced by M.

Let $M' \in \mathbb{Z}_\omega^{Q\times Q}$ and $P_0, R_1 \subseteq Q$. Let $P_1 = P_0 \cdot \alpha(M')$, $R_0 = \alpha(M') \cdot R_1$, and let M be the restriction of M' to $(P_0 \cap R_0) \times (P_1 \cap R_1)$. We denote $\left(\begin{smallmatrix} P_0 \\ R_0 \end{smallmatrix}\ M\ \begin{smallmatrix} P_1 \\ R_1 \end{smallmatrix}\right)$ by $[\![P_0, M', R_1]\!]$ and call it the *metatransition induced by* P_0, M', R_1. We also say that $[\![P_0, M', R_1]\!]$ is induced by M'.

Lemma 3.1. *Let* $t_1 = \left(\begin{smallmatrix} P_0 \\ R_0 \end{smallmatrix}\ M_1\ \begin{smallmatrix} P_1 \\ R_1 \end{smallmatrix}\right)$, $t_2 = \left(\begin{smallmatrix} P_1 \\ R_1 \end{smallmatrix}\ M_2\ \begin{smallmatrix} P_2 \\ R_2 \end{smallmatrix}\right) \in \mathsf{MT}(\mathbb{Z}_\omega, Q)$ *and* $M_1', M_2' \in \mathbb{Z}_\omega^{Q\times Q}$. *If* $t_1 = [\![P_0, M_1', R_1]\!]$ *and* $t_2 = [\![P_1, M_2', R_2]\!]$, *then* $t_1 t_2 = [\![P_0,\ M_1'M_2',\ R_2]\!]$.

Let $k \geq 1$ and let $M_1', \ldots, M_k' \in \mathbb{Z}_\omega^{Q\times Q}$. Let $P_0, R_k \subseteq Q$. As above, the matrices $M_1', \ldots, M_k'$ induce with P_0, R_k a sequence of concatenable metatransitions: For $0 < i \leq k$, let $P_i = P_{i-1} \cdot \alpha(M_i')$ and $R_{i-1} = \alpha(M_i') \cdot R_i$. Finally let M_i be the restriction of M_i' to $(P_{i-1} \cap R_{i-1}) \times (P_i \cap R_i)$ and $t_i = \left(\begin{smallmatrix} P_{i-1} \\ R_{i-1} \end{smallmatrix}\ M_i\ \begin{smallmatrix} P_i \\ R_i \end{smallmatrix}\right)$ for $1 \leq i \leq k$.

Clearly, $t_i = [\![P_{i-1}, M_i', R_i]\!]$. Moreover, $P_i \cap R_i \neq \emptyset$ for some $0 \leq i \leq k$ if and only if $P_i \cap R_i \neq \emptyset$ for every $0 \leq i \leq k$. By Lemma 3.1, we obtain $t_1 \cdots t_k = [\![P_0,\ M_1 \cdots M_k,\ R_k]\!]$.

3.2. The Semigroup of Metatransitions of an Automaton

Let $\mathcal{A} = [Q, \mu, \lambda, \varrho]$ be a min-plus automaton, $I = \{\, q \in Q \mid \lambda[q] \in \mathbb{Z} \,\}$ and $F = \{\, q \in Q \mid \varrho[q] \in \mathbb{Z} \,\}$.

Let $n \geq 1$ and $w_1, \ldots, w_n \in \Sigma^*$ be a sequence of words. As above, the matrices $\mu(w_1), \ldots, \mu(w_n)$ induce with $P_0 = I$ and $R_n = F$ a sequence of concatenable metatransitions $t_1, \ldots, t_n$.

Let $q_0, \ldots, q_n \in Q$. If $\lambda[q_0] + \big(\mu(w_1), \ldots, \mu(w_n)\big)[q_0, \cdots, q_n] + \varrho[q_n] \in \mathbb{Z}$, then we have $q_i \in P_i \cap R_i$ for every $0 \leq i \leq k$. Conversely, for every $1 \leq i \leq k$ and every $q \in P_i \cap R_i$, $\mathcal{A}$ can read $w_1 \cdots w_i$ from an initial state to q, and it can read $w_{i+1} \cdots w_n$ from q to an accepting state. In this sense, the metatransitions $t_1, \ldots, t_n$ represent exactly the accepting paths for w in $\mathcal{A}$. The matrices inside $t_1, \ldots, t_n$ are the matrices $\mu(w_1), \ldots, \mu(w_n)$ restricted to the entries which occur in accepting paths for $w_1 \ldots w_n$.

We have $P_i \cap R_i \neq \emptyset$ for some $0 \leq i \leq k$ if and only if $P_i \cap R_i \neq \emptyset$ for every $0 \leq i \leq k$ if and only if $\mathcal{A}$ accepts $w_1 \ldots w_n$.

The most beautiful property is the following: let $0 \leq i < j \leq n$ and assume $P_i = P_j$ and $R_i = R_j$. We consider the sequence of words $w' = w_1, \ldots, w_i, (w_{i+1}, \ldots, w_j,)^k w_{j+1}, \ldots, w_n$ for some $k \geq 0$. By applying μ to each word in w', we obtain a sequence of matrices. As above, these matrices induce with $P_0 = I$ and $R_n = F$ a sequence of metatransitions. Clearly, we obtain the sequence $t_1, \ldots, t_i, (t_{i+1}, \ldots, t_j,)^k t_{j+1}, \ldots, t_{|w|}$.

Although this property looks quite obvious, it is of crucial importance since it enables us to apply pumping- and Burnside-techniques.

We associate to $\mathcal{A}$ a subsemigroup of $\mathsf{MT}(\mathbb{Z}_\omega, Q)_0$. We call some set $S \subseteq Q$ a *P-clone of* $\mathcal{A}$ (resp. an *R-clone of* $\mathcal{A}$) if there exists some word $v \in \Sigma^*$ such that $S = I \cdot \alpha(\mu(v))$ (resp. $S = \alpha(\mu(v)) \cdot F$). Let $\mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A}) =$

$$\Big\{ [\![P_0, \mu(a), R_1]\!] \,\Big|\, a \in \Sigma, P_0 \text{ is a P-clone}, R_1 \text{ is a R-clone}, P_0 \cdot \mu(a) \cdot R_1 \neq \infty \Big\}.$$

The condition $P_0 \cdot \mu(a) \cdot R_1 \neq \infty$ ensures that $\mu(a)$ does not restrict to a $\emptyset \times \emptyset$-matrix.

Let $\langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A}) \rangle_0$ be the subsemigroup of $\mathsf{MT}(\mathbb{Z}_\omega, Q)_0$ generated by $\mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A}) \cup \{0\}$. By Lemma 3.1, we can show that $\langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A}) \rangle_0$ consists of 0 and metatransitions of the form $[\![P_0, \mu(w), R_1]\!]$ for P-clones P_0, R-clones R_1, and words $w \in \Sigma^+$ satisfying $P_0 \cdot \mu(w) \cdot R_1 \neq \infty$.

For every metatransition $t_2 \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0$, there are $t_1, t_3 \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0$ such that $t_1t_2t_3 \neq 0$ and $t_1t_2t_3 = [\![I, \mu(w), F]\!]$ for some word $w \in \Sigma^*$.

By removing the weights from $\mathcal{A}$, we can define in the same way a set $\mathsf{MT}(\mathbb{B}, \mathcal{A})$ and the subsemigroup $\langle \mathsf{MT}(\mathbb{B}, \mathcal{A})\rangle_0$ of $\mathsf{MT}(\mathbb{B}, \mathcal{A})$.

3.3. On Metatransitions with an Idempotent Structure

Let $\mathcal{A}$ be a polynomially ambiguous min-plus automaton and let $e = \left(\begin{smallmatrix} P \\ R \end{smallmatrix} M \begin{smallmatrix} P \\ R \end{smallmatrix}\right) \in \langle \mathsf{MT}(\mathbb{Z}_\omega, Q)\rangle_0$ be a metatransition with an *idempotent structure*, i.e., assume $\alpha(ee) = \alpha(e)$. We define a relation $\leq_e$ on $P \cap R$ by setting $p \leq_e q$ iff $M[p, q] \neq \infty$. This relation is "almost a partial order" in that it satisfies the three following properties.

i) Clearly, $\leq_e$ is transitive.

ii) The relation $\leq_e$ is antisymmetric. Let $p \neq q \in P \cap R$ such that $p \leq_e q \leq_e p$. Clearly, e is induced by $\mu(v)$ for some $v \in \Sigma^*$. Then, $\mathcal{A}$ can read v^2 from p to p in two paths. One path stays at p, the other path goes from p to q and back. This contradicts Theorem 2.1.

iii) For every $q \in P \cap R$, there exist $p, r \in P \cap R$, $p \leq_e q \leq_e r$ such that $p \leq_e p$ and $r \leq_e r$. By ($\mathsf{MT}3$), there are $q_1, q_2, \ldots \in P \cap R$ such that $q \leq_e q_1 \leq_e q_2 \leq_e \cdots$ By transitivity and finiteness of $P \cap R$, there is some r among $q_1, q_2, \ldots$ such that $q \leq_e r \leq_e r$. The proof for p is similar.

Lemma 3.2. *Let $e = \left(\begin{smallmatrix} P \\ R \end{smallmatrix} M \begin{smallmatrix} P \\ R \end{smallmatrix}\right) \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0$ with an idempotent structure. Let $k \geq 1$ and $p, q \in P \cap R$ such that $M^k[p, q] \neq \infty$.*

There are $p = p_0, \ldots, p_k = q$ in $P \cap R$ such that $M^k[p, q] = M[p_0, \ldots, p_k]$. Moreover, if $k > |P \cap R|$, then we can choose $p_0, \ldots, p_k$ such that there are $0 \leq i < j \leq k$ such that $p_i = p_{i+1} = \cdots = p_j$ and $p_0, \ldots, p_i, p_{j+1}, \ldots, p_k$ does not contain a cycle.

The last claim of Lemma 3.2 just says that for large k, one can choose the sequence $p_0, \ldots, p_k$ to be almost constant up to a short cycle-free prefix and suffix. The total length of the prefix and the suffix is at most $|P \cap R|$.

It is important that for the antisymmetry of $\leq_e$ and for Lemma 3.2, we do not need to assume $e \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0$, it suffices that $e \in \mathsf{MT}(\mathbb{Z}_\omega, Q)$ and $\alpha(e) = \alpha(ee) \in \langle \mathsf{MT}(\mathbb{B}, \mathcal{A})\rangle_0$.

3.4. Stabilization

Let $\mathcal{A}$ be a polynomially ambiguous min-plus automaton. Let $e = \left(\begin{smallmatrix} P \\ R \end{smallmatrix} M \begin{smallmatrix} P \\ R \end{smallmatrix}\right) \in \mathsf{MT}(\mathbb{Z}_\omega, Q)$ such that $\alpha(e) = \alpha(ee) \in \langle \mathsf{MT}(\mathbb{B}, \mathcal{A})\rangle_0$. Assume $\mathsf{mind}(M) = 0$.

We define $M^\sharp$, the *stabilization of* M. The idea of $M^\sharp$ is to understand the sequence $(M^k)_{k\geq 1}$. Let $p, q \in P \cap R$.

If $M^k[p, q] = \infty$, for some $k \geq 1$, then $M^k[p, q] = \infty$ for every $k \geq 1$. In this case, we define $M^\sharp[p, q] = \infty$.

Assume $M[p, q] \neq \infty$. Lemma 3.2 is crucial to understand the sequence $(M^k[p, q])_{k\geq 1}$. From $\mathsf{mind}(M) = 0$, we can easily deduce a lower bound on $(M^k[p, q])_{k\geq 1}$.

We say that some sequence $p_0, \ldots, p_k \in P \cap R$ satisfies (S1), if $p_0 = p$, $p_k = q$, and $M[p_0, \ldots, p_k] \in \mathbb{Z}$. If $p_0, \ldots, p_k$ satisfies (S1) and there exists some $0 \leq i \leq k$ such that $M[p_i, p_i] = 0$, then we say that $p_0, \ldots, p_k$ satisfies (S2).

Assume there exists a sequence which satisfies (S2). Then, there exists a sequence $p_0, \ldots, p_k$ for some $k < |P \cap R|$ which satisfies (S2) such that $m = M[p_0, \ldots, p_k]$ is minimal

among all sequences which satisfy (S2). In this case, $(M^k[p,q])_{k\geq 1}$ is ultimately constant m and we define $M^\sharp[p,q] = m$.

Assume that there does not exist a sequence which satisfies (S2) although $M[p,q] \neq \infty$. We can conclude that the sequence $(M^k[p,q])_{k\geq 1}$ is either ultimately ω, or it tends to infinity, since (S1)-sequences cannot utilize the zeros on the main diagonal of M. In this case, we set $M^\sharp[p,q] = \omega$.

Consequently, $M^\sharp[p,q]$ describes the behaviour of $(M^k[p,q])_{k\geq 1}$.

For $p \in P \cap R$ satisfying $M[p,p] = 0$, we have $M^\sharp[p,p] = 0$.

We generalize the definition of $M^\sharp$ by weakening the assumption $\mathsf{mind}(M) = 0$ to $\mathsf{mind}(M) \in \mathbb{Z}$. We still assume $\alpha(e) = \alpha(ee) \in \langle \mathsf{MT}(\mathbb{B}, \mathcal{A})\rangle_0$.

We normalize M. Let $m = \mathsf{mind}(M)$ and define $\bar{M}$ by $\bar{M}[p,q] = M[p,q] - m$ for[2] $p, q \in P \cap R$. Clearly, $\alpha(\bar{M}) = \alpha(M)$ and $\mathsf{mind}(\bar{M}) = 0$. We define $M^\sharp = \bar{M}^\sharp$.

For $k \geq 1$, we have $M^k[p,q] = km + \bar{M}^k[p,q]$. Let $p, q, p', q' \in P \cap R$. For $k \geq 1$, we have $M^k[p,q] - M^k[p',q'] = \bar{M}^k[p,q] - \bar{M}^k[p',q']$, provided that $M^k[p,q] \in \mathbb{Z}$.

If $M^\sharp[p,q]$ and $M^\sharp[p',q']$ are integers, then the entries $[p,q]$ and $[p',q']$ are ultimately constant in $(\bar{M}^k)_{k\geq 1}$, i.e., the entries $[p,q]$ and $[p',q']$ grow or sink synchronized in the sequence $(M^k)_{k\geq 1}$ and for every k beyond some bound, we have $M^k[p,q] - M^k[p',q'] = M^\sharp[p,q] - M^\sharp[p',q']$.

However, if $M^\sharp[p,q] = \omega$ and $M^\sharp[p',q'] \in \mathbb{Z}$, then either $(M^k[p,q])_{k\geq 1}$ is ultimately ω, or the difference $(M^k[p,q] - M^k[p',q'])_{k\geq 1}$ tends to infinity.

Given $e = \left(\begin{smallmatrix} P \\ R \end{smallmatrix}\ M\ \begin{smallmatrix} P \\ R \end{smallmatrix}\right) \in \mathsf{MT}(\mathbb{Z}_\omega, Q)$ satisfying $\alpha(e) = \alpha(ee) \in \langle \mathsf{MT}(\mathbb{B}, \mathcal{A})\rangle_0$ and $\mathsf{mind}(M) \in \mathbb{Z}$, we define its *stabilization* $e^\sharp = \left(\begin{smallmatrix} P \\ R \end{smallmatrix}\ M^\sharp\ \begin{smallmatrix} P \\ R \end{smallmatrix}\right) \in \mathsf{MT}(\mathbb{Z}_\omega, Q)$. We have $\alpha(e^\sharp) = \alpha(e)$ and $e^\sharp \in \mathsf{MT}(\mathbb{Z}_\omega, Q)$.

Finally, let $t \in \mathsf{MT}(\mathbb{Z}_\omega, Q)$ and let M be the matrix in t. We define $\mathsf{span}(t) = \mathsf{span}(M)$ and generalize the notions min, max, and mind in the same way.

Lemma 3.3. (1) *For concatenable $t_1, t_2 \in \mathsf{MT}(\mathbb{Z}, Q)$, $\mathsf{span}(t_1 t_2) \leq \mathsf{span}(t_1) + \mathsf{span}(t_2)$.*
(2) *For $e \in \mathsf{MT}(\mathbb{Z}_\omega, Q)$ for which $e^\sharp$ is defined, we have $\mathsf{span}(e^\sharp) \leq |Q|\mathsf{span}(e)$.*

3.5. Main Results, Conclusions, and Open Questions

Let $\mathcal{A}$ be a polynomially ambiguous min-plus automaton and let $\mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})$ as in Section 3.2. Let $\langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$ be the least semigroup which

(1) contains $\mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})$ and the zero of $\mathsf{MT}(\mathbb{Z}_\omega, Q)$,
(2) is closed under the product of metatransitions, and
(3) is closed under stabilization, i.e., for every $e \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$, we have $e^\sharp \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$, provided that $e^\sharp$ is defined.

We state our main characterization:

Theorem 3.4. *Let $\mathcal{A}$ be a polynomially ambiguous min-plus automaton. The following assertions are equivalent:*

(1) *There exists some metatransition $t \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$ such that every entry in t belongs to $\{\omega, \infty\}$.*
(2) *The min-plus automaton $\mathcal{A}$ has no unambiguous equivalent.*

[2]Whereas $\infty - m = \infty$ and $\omega - m = \omega$.

(3) *Every unambiguous min-plus automaton $\tilde{\mathcal{A}}$ which accepts the same language as $\mathcal{A}$ satisfies one of the following conditions:*
   (3a) *There are $u, v, w \in \Sigma^*$ such that $uv^k w$ is accepted by $\mathcal{A}$ and $\tilde{\mathcal{A}}$ for $k \geq 1$, and for growing k, the sequence $\big(|\tilde{\mathcal{A}}|(uv^k w) - |\mathcal{A}|(uv^k w)\big)_{k\geq 1}$ tends to infinity.*
   (3b) *There is a $\sharp$-expression r such that $r(k)$ is accepted by $\mathcal{A}$ and $\tilde{\mathcal{A}}$ for $k \geq 1$, and for growing k, the sequence $\big(|\mathcal{A}|(r(k)) - |\tilde{\mathcal{A}}|(r(k))\big)_{k\geq 1}$ tends to infinity.*

The reader might complain that (as seen in Section 3.4) one entry in the main diagonal of a stabilization $e^\sharp$ is 0, and hence, some matrix t as in Theorem 3.4(1) cannot exist. However, by applying both stabilization and multiplication, metatransitions in which every entry is either ω or ∞ may arise.

For illustration, let us consider Theorem 3.4 for the particular case that $\mathcal{A}$ is unambiguous. Let $e = \left(\begin{smallmatrix} P \\ R \end{smallmatrix}\ M\ \begin{smallmatrix} P \\ R \end{smallmatrix}\right) \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$ be with an idempotent structure. Since P (resp. R) is a P- (resp. R-clone), there are $u, v \in \Sigma^*$ such that $P = I \cdot \alpha(\mu(u))$ and $R = \alpha(\mu(v)) \cdot F$. If $|P \cap R| > 1$, then we can construct two different accepting paths for uv. Hence, $P \cap R = 1$ and M is a (1×1)-matrix. By (MT3), the entry of M cannot be ∞. If the entry of M is an integer, then $\mathsf{mind}(M)$ yields the only entry of M, and thus, the entry of the normalization $\bar{M}$ is 0, i.e., the entry of $M^\sharp = \bar{M}^\sharp$ is 0. Consequently, ω's cannot arise in the closure $\langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$, and in particular, (1) in Theorem 3.4 is not satisfied.

Note that (3) $\Rightarrow$ (2) in Theorem 3.4 is obvious. We will prove (1) $\Rightarrow$ (3) in Section 4. We assume some t as in (1) and assume some $\tilde{\mathcal{A}}$ as in (3) which does not satisfy (3a). Then, we show (3b): as t is constructed from metatransitions in $\mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})$ by using multiplication and stabilization, r is constructed from letters by using concatenation and $\sharp$-powers.

We will prove (2) $\Rightarrow$ (1) in Section 5. It leads to an intriguing combination of two Burnside problems over metatransitions which are remotely related to problems considered by Simon and Leung, e.g. [18, 23, 24].

Theorem 3.5. *Given a polynomially ambiguous min-plus automaton $\mathcal{A}$, we can decide whether $\mathcal{A}$ has an unambiguous equivalent, or whether it has a sequential equivalent.*

Proof. To decide the existence of an unambiguous equivalent, one process searches for some $t \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$ as in Theorem 3.4(1). A simultaneous process enlists all unambiguous min-plus automata, and checks (using an algorithm in [17]) whether one of them is equivalent to $\mathcal{A}$. By Theorem 3.4, exactly one of the processes terminates. To decide the existence of a sequential equivalent, the algorithm decides at first whether there exists an unambiguous equivalent $\mathcal{A}'$. If so, it applies an algorithm in [15, 21] to $\mathcal{A}'$. ∎

It is interesting to have by Theorem 3.5 a decidability result for a class of min-plus automata for which the equivalence problem is undecidable [16]. Many interesting questions arise from our approach and from the introduced proof techniques. The central question is of course whether or how our approach can be generalized to arbitrary min-plus automata. Another question is whether we can achieve complexity results or a practical algorithm.

Further questions are: can we characterize the existence of a sequential equivalent in terms of the stabilization closure $\langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$? Is the existence of a finitely ambiguous (resp. finitely sequential) equivalent decidable? Is the membership problem of $\langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$ decidable? Are our techniques helpful to decide the open equivalence problem between a polynomially and a finitely ambiguous min-plus automaton?

4. Necessity

We prove (1) $\Rightarrow$ (3) in Theorem 3.4. We assume some polynomially ambiguous min-plus automaton $\mathcal{A} = [Q, \mu, \lambda, \varrho]$ which satisfies (1). We assume an unambiguous automaton $\tilde{\mathcal{A}} = [\tilde{Q}, \tilde{\mu}, \tilde{\lambda}, \tilde{\varrho}]$ which accepts the same language and show (3).

Since $\mathcal{A}$ satisfies Theorem 3.4(1), there exists some $s \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$ such that every entry in s is ω or ∞. We can assume that s is of the form $s = \left(\begin{smallmatrix} I \\ F_s \end{smallmatrix}\ M\ \begin{smallmatrix} I_s \\ F \end{smallmatrix}\right)$ for some $F_s, I_s \subseteq Q$ and some M. Since $\alpha(s) \in \langle \mathsf{MT}(\mathbb{B}, \mathcal{A})\rangle_0$, we have $I \cap F_s \neq \emptyset$ and $I_s \cap F \neq \emptyset$.

To explain the idea, let us assume that s is of the form $s = t_1 e_2^\sharp t_3 e_4^\sharp t_5$ for some metatransitions $t_1, e_2, t_3, e_4, t_5 \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0$, i.e., there is no ω in t_1, e_2, t_3, e_4, t_5. Let $u_1, \ldots, u_5 \in \Sigma^*$ such that t_1, e_2, t_3, e_4, t_5 are induced by $\mu(u_1), \ldots, \mu(u_5)$ with I and F. We denote by $M_1, \ldots, M_5$ the matrices inside t_1, e_2, t_3, e_4, t_5.

Let ℓ be some extremely large multiple of $|\tilde{Q}|!$. We show that (3a) or (3b) is satisfied.

At first, we consider the output of $\mathcal{A}$ on words $u_1 u_2^{\ell k} u_3 u_4^{\ell} u_5$ for large, growing k. The output of $\mathcal{A}$ on such words for large k is mainly determined by $u_2^{\ell k}$. It should be clear that for large growing k, the output of $\mathcal{A}$ grows by $\ell \cdot \mathsf{mind}(M_2)$ per k, i.e., the growth rate is $\ell \cdot \mathsf{mind}(M_2)$ per k.

Similarly, the output of $\mathcal{A}$ on $u_1 u_2^{\ell} u_3 u_4^{\ell k} u_5$ for large, growing k has a growth rate of $\ell \cdot \mathsf{mind}(M_4)$ per k.

However, what happens for words $u_1 u_2^{\ell k} u_3 u_4^{\ell k} u_5$ for large, growing k. Assume the growth rate of the output of $\mathcal{A}$ on this sequence is $\ell \cdot (\mathsf{mind}(M_2) + \mathsf{mind}(M_4))$. Assume some extremely large k and consider some accepting path π for the word $u_1 u_2^{\ell k} u_3 u_4^{\ell k} u_5$. Assume that the weight of π yields $|\mathcal{A}|(u_1 u_2^{\ell k} u_3 u_4^{\ell k} u_5)$. We decompose π into $\pi_1, \ldots, \pi_5$ which correspond to $u_1, u_2^{\ell k}, u_3, u_4^{\ell k}, u_5$, and denote the first and last states of $\pi_1, \ldots, \pi_5$ by $i_0, \ldots, i_5$. For example π_2 starts in i_1, ends in i_2 and reads $u_2^{\ell k}$.

To achieve the growth rate of $\ell \cdot (\mathsf{mind}(M_2) + \mathsf{mind}(M_4))$ per k, $\mathcal{A}$ has to read almost every u_2 with a weight of $\mathsf{mind}(M_2)$, and has read almost every u_4 with a weight of $\mathsf{mind}(M_4)$. Hence, the paths π_2 and π_4 have to utilize the least entries on the main diagonal on M_2 and M_4, respectively.

We can factorize π_2 into ℓk factors such that each factor reads u_2. Let us denote by $r_0, \ldots, r_{\ell k}$ the first and last states of these factors, in particular, $i_1 = r_0$ and $r_{\ell k} = i_2$. Since, π_2 utilizes a least entry on the main diagonal of M_2, $r_0, \ldots, r_{\ell k}$ utilize a 0 on the main diagonal of the normalization $\bar{M}_2$, i.e., $r_0, \ldots, r_{\ell k}$ satisfy (S2). Hence, $M_2^\sharp[i_1, i_2] = \bar{M}_2^\sharp[i_1, i_2] \in \mathbb{Z}$. By the same argument, we obtain $M_4^\sharp[i_3, i_4] \in \mathbb{Z}$. Consequently, $s[i_0, i_5] \leq (M_1, M_2^\sharp, M_3, M_4^\sharp, M_5)[i_0, \ldots, i_5] \in \mathbb{Z}$, i.e., $s[i_0, i_5] \in \mathbb{Z}$ which contradicts the choice of s.

Consequently, the growth rate of the output of $\mathcal{A}$ on words $u_1 u_2^{\ell k} u_3 u_4^{\ell k} u_5$ for large, growing k is strictly larger than $\ell \cdot (\mathsf{mind}(M_2) + \mathsf{mind}(M_4))$.

Next, we analyze how $\tilde{\mathcal{A}}$ reads $u_1 u_2^{\ell} u_3 u_4^{\ell} u_5$. Let π be the unique accepting path of $u_1 u_2^{\ell} u_3 u_4^{\ell} u_5$ in $\tilde{\mathcal{A}}$. As above, we decompose π into $\pi_1, \ldots, \pi_5$ which correspond to $u_1, u_2^{\ell}, u_3, u_4^{\ell}, u_5$, and denote the first and last states of $\pi_1, \ldots, \pi_5$ by $i_0, \ldots, i_5$.

For the structure of π_2 and π_4, Lemma 2.2 is very helpful. Since ℓ is extremely larger than $|\tilde{Q}|$, π_2 consists mainly of a short cycle π_2' which is looped many times. Let n_2 be the number of u_2's which are read in this cycle. Let m_2 the weight of π_2' divided by n_2. The value m_2 can be understood as the relative cycle weight of π_2.

Now, we consider the output of $\tilde{\mathcal{A}}$ on words $u_1u_2^{\ell k}u_3u_4^{\ell}u_5$ for large, growing k. Since the factors u_2 are read in many looped π_2' cycles, the growth rate of the output of $\tilde{\mathcal{A}}$ is $\ell k m_2$ per k.

By applying the same argument on π_4 we obtain some m_4, and the growth rate of the output of $\tilde{\mathcal{A}}$ on words $u_1u_2^{\ell}u_3u_4^{\ell k}u_5$ for large, growing k is $\ell k m_4$ per k.

Since $\tilde{\mathcal{A}}$ is unambiguous, the growth rate of the output of $\tilde{\mathcal{A}}$ on words $u_1u_2^{\ell k}u_3u_4^{\ell k}u_5$ for large, growing k is $\ell k(m_2+m_4)$ per k.

Now, at least one of the following three cases occurs:

- $km_2 > \mathsf{mind}(M_2)$ Then, on words $u_1u_2^{\ell k}u_3u_4^{\ell}u_5$ for growing k, the output of $\tilde{\mathcal{A}}$ grows faster than the output of $\mathcal{A}$. Hence, we have (3a) by using u_1, u_2^{ℓ}, $u_3u_4^{\ell}u_5$ as u, v, w.
- $km_4 > \mathsf{mind}(M_4)$ Like the previous case.
- $km_2 \leq \mathsf{mind}(M_2)$ and $km_4 \leq \mathsf{mind}(M_4)$ We consider words $u_1u_2^{\ell k}u_3u_4^{\ell k}u_5$ for growing k. The growth rate of $\mathcal{A}$ on these words is strictly larger than $\ell \cdot (\mathsf{mind}(M_2) + \mathsf{mind}(M_4))$ per k, whereas the growth rate of $\tilde{\mathcal{A}}$ is less than $\ell k(m_2+m_4)$ per k. Hence, we have (3b) by using $u_1(u_2^{\ell})^{\sharp}u_3(u_4^{\ell})^{\sharp}u_5$ as r.

Thus, we have shown (3) in the particular case that s is of the form $t_1e_2^{\sharp}t_3e_4^{\sharp}t_5$. It is straightforward to generalize this argument for s which are of the form $t_1e_2^{\sharp}t_3\dots e_{n-1}^{\sharp}t_n$ for some n. However, this generalization is not sufficient. The real technical challenge is to prove (3) for some s which is generated by nesting stabilizations, e.g., if s is of the form $t_1\big(e_2^{\sharp}t_3e_4^{\sharp}\big)^{\sharp}t_5$ or if s is generated by arbitrarily many nested stabilizations.

To deal with these cases, we have to develop the same argumentation as above in a tree-like fashion. As above, we assume by Theorem 3.4(1) some $s \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^{\sharp}$ which is of the form $s = \left(\begin{smallmatrix} I & & I_s \\ F_s & M & F \end{smallmatrix}\right)$ whereas every entry in M belongs to $\{\omega, \infty\}$.

We define the notion of a $\sharp$-tree. Its nodes are labeled with triples (w,t,t') whereas $w \in \Sigma^*$, $t \in \langle \mathsf{MT}(\mathbb{Z}, \mathcal{A})\rangle_0^{\sharp}$, and $t' \in \langle \mathsf{MT}(\mathbb{Z}, \mathcal{A})\rangle_0$, satisfying $\alpha(t) = \alpha(t')$.

Let $k \geq 1$. We define now $\sharp$-trees of rank k. For every $a \in \Sigma$, every P-clone P and every R-clone R, there is a $\sharp$-*tree* which consists of a single node labeled with $\big(a,t,t\big)$, whereas $t = [\![P, \mu(a), R]\!]$.

Let T_1, T_2 be $\sharp$-trees and assume that their roots are labeled with (w_1,t_1,t_1') and (w_2,t_2,t_2'), respectively. If t_1 and t_2 are concatenable, then we construct a $\sharp$-*tree* as follows: its root is labeled with $(w_1w_2,\, t_1t_2,\, t_1't_2')$. Its successors are T_1 and T_2.

Let T_1 be a $\sharp$-tree and assume that its root is labeled with (w_1,t_1,t_1'). If $t_1^{\sharp}$ is defined, then we construct another $\sharp$-*tree*: its root is labeled with $(w_1^k, t_1^{\sharp}, t_1'^k)$ and has k copies of T_1 as successors.

For every $t \in \langle \mathsf{MT}(\mathbb{Z}, \mathcal{A})\rangle_0^{\sharp}$, there are some $w \in \Sigma^*$, $t' \in \langle \mathsf{MT}(\mathbb{Z}, \mathcal{A})\rangle_0$, and a $\sharp$-tree whose root is labeled with (w,t,t').

Consequently, there are $w \in \Sigma^*$, $s' \in \langle \mathsf{MT}(\mathbb{Z}, \mathcal{A})\rangle_0$, and a $\sharp$-tree whose root is labeled with (w,s,s'). We can naturally associate a $\sharp$-expression r to this $\sharp$-tree in a bottom-up-manner, and we have $r(k) = w$.

We can then prove various conditions in a bottom-up induction over the nodes of the $\sharp$-tree. The key argumentation is as follows: we assume that w does not admit a factorization into three words which prove (3a). Under this assumption, we can show that $|\mathcal{A}|(w)$ is much larger than $|\tilde{\mathcal{A}}|(w)$, and we can show in particular that r can be used to prove (3b).

5. Sufficiency

We show (2) $\Rightarrow$ (1) in Theorem 3.4 by contraposition. We assume a polynomially ambiguous min-plus automaton $\mathcal{A} = [Q, \mu, \lambda, \varrho]$ which does not satisfy (1), and we construct an equivalent unambiguous automaton. We assume that the entries of λ and ϱ are 0 or ∞.

The construction of an unambiguous equivalent relies on the following proposition:

Proposition 5.1. *Let $\mathcal{A}$ be a polynomially ambiguous min-plus automaton, and assume that $\mathcal{A}$ does not satisfy (1) in Theorem 3.4.*

There is some $Y \geq 0$ such that the following assertion is true:

For every $t = \left(\begin{smallmatrix} P \\ R \end{smallmatrix} M \begin{smallmatrix} P' \\ R' \end{smallmatrix}\right) \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0$, there is some $t' = \left(\begin{smallmatrix} P \\ R \end{smallmatrix} M' \begin{smallmatrix} P' \\ R' \end{smallmatrix}\right) \in \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp$ satisfying:

(A1): $\alpha(t) = \alpha(t')$

(A2): *For every $p \in P \cap R$, $q \in P' \cap R'$, satisfying $M[p,q] \neq \infty$ and*

$$M[p,q] \;\geq\; \mathsf{min}(M) + Y, \qquad \text{we have} \qquad M'[p,q] = \omega.$$

The proof of Proposition 5.1 leads us to an intriguing combination of two Burnside problems for metatransitions. The main proof of Proposition 5.1 utilizes an inductive argument via the factorization forest theorem for the homomorphism $\alpha : \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp \to \langle \mathsf{MT}(\mathbb{B}, \mathcal{A})\rangle_0$. The induction step for metatransitions with an idempotent structure leads us to another Burnside problem itself. To solve this inner Burnside problem, we consider subsemigroups $T_e = \langle \mathsf{MT}(\mathbb{Z}_\omega, \mathcal{A})\rangle_0^\sharp \cap \alpha^{-1}(e)$ for idempotents $e \in \langle \mathsf{MT}(\mathbb{B}, \mathcal{A})\rangle_0$. This inner Burnside problem is then shown by methods which are remotely related to techniques by Simon and Leung for the limitedness problem of distance automata [18, 23, 24].

To prove Proposition 5.1 by an induction via the factorization forest theorem, we have to add two more technical conditions to get a stronger inductive hypothesis.

One can deduce Y from the proof of Proposition 5.1. It is elementary but superexponential. Knowing Y is not required to show the decidability in Theorem 3.5.

We construct now an unambiguous equivalent $\mathcal{A}'$ of $\mathcal{A}$.

For every R-clone R satisfying $R \cap I \neq \emptyset$, we add an initial state (I, δ, R) to $\mathcal{A}'$, whereas δ is a $(0, \ldots, 0)$ tuple of dimension $I \cap R$.

Next, we construct for every state of $\mathcal{A}'$ the outgoing transitions and the follow state.

Let (P, δ, R) be some already constructed state of $\mathcal{A}'$. For every $a \in \Sigma$ and every R-clone R' satisfying $R = \alpha(\mu(a)) \cdot R'$, we add a transition and a state to $\mathcal{A}$ as follows:

(1) Let $t = \left(\begin{smallmatrix} P \\ R \end{smallmatrix} M \begin{smallmatrix} P' \\ R' \end{smallmatrix}\right)$ be the metatransition induced by $\mu(a)$ with P and R'.

(2) Let $\hat{\delta} = \delta \cdot M$. Hence, $\hat{\delta}$ is a tuple of dimension $(P' \cap R')$.

(3) We normalize $\hat{\delta}$. For every $q \in P' \cap R'$, we set $\delta'[q] = \hat{\delta}[q] - \mathsf{min}(\hat{\delta})$.

(4) We introduce a transition from (P, δ, R) to (P', δ', R').

(5) The label and the weight of this transition are a and $\mathsf{min}(\hat{\delta})$, respectively.

In this way, we can construct the entire min-plus automaton $\mathcal{A}'$. At this point of the construction, the set of states might become infinite.

Some state (P, δ, R) is an accepting state if $R = F$. The accepting weight is 0.

Consider some word $w = a_1 \ldots a_n \in \Sigma^*$ which is accepted by $\mathcal{A}'$. Denote the states of the accepting path for w in $\mathcal{A}'$ by (P_i, δ_i, R_i) for $i \in \{0, \ldots, n\}$. In particular, $P_0 = I$ and $R_n = F$. For $i \in \{1, \ldots, n\}$, denote by m_i the transition weight of the i-th transition of π.

Let $1 \leq i \leq n$. By an induction on i, we can show that for every $q \in P_i \cap R_i$, the sum $m_1 + \cdots + m_i + \delta_i[q]$ is exactly $(I \cdot \mu(a_1 \dots a_i))[q]$. The sum $m_1 + \cdots + m_n$ is then the minimum of $(I \cdot \mu(w))[q]$ for $q \in F$, i.e., the sum $m_1 + \cdots + m_n$ is $\lambda\mu(w)\varrho = \mathcal{A}(w)$.

Conversely, consider some word $w = a_1 \dots a_n \in \Sigma^*$ which is accepted by $\mathcal{A}$. We can construct an accepting path for w in $\mathcal{A}'$ as follows. Let $t_1, \dots, t_n$ be the metatransitions induced by $\mu(a_1), \dots, \mu(a_n)$ with I and F. Denote $t_i = \left(\begin{smallmatrix} P_{i-1} & & P_i \\ R_{i-1} & M_i & R_i \end{smallmatrix}\right)$. The state (P_0, δ, R_0) (whereas δ is the $(0, \dots, 0)$ tuple of dimension $P_0 \cap R_0$) is the first state of the constructed path. Then, we proceed along the above steps (1) to (5) for each a_i and each R_i for $i \in \{1, \dots, n\}$ and obtain an accepting path for w in $\mathcal{A}'$. We can apply the above argumentation to show that the sum of the transition weights is exactly $\lambda\mu(w)\varrho = \mathcal{A}(w)$.

Consequently, $|\mathcal{A}|$ and $|\mathcal{A}'|$ are equivalent, and it is easy to verify that $\mathcal{A}'$ is unambiguous. However, a major problem remained: we cannot show that $\mathcal{A}'$ has finitely many states. We overcome this problem by changing step (3) in the construction above as follows:

(3') We normalize $\hat{\delta}$. For every $q \in P' \cap R'$, we set $\delta''[q] = \hat{\delta}[q] - \mathsf{min}(\hat{\delta})$. Then, we construct δ' by replacing in δ'' every non-∞ entry which is larger than $2Y$ by ω.

By using (3') instead of (3), the set of states of $\mathcal{A}'$ will be finite. We have to show that the construction of $\mathcal{A}'$ is still correct, that is that every entry that becomes too large can be replaced by ω.

Let $u_1, u_2 \in \Sigma^*$ and assume that $\mathcal{A}$ accepts u_1u_2. Let $I = \{\, q \in Q \mid \lambda[q] \in \mathbb{Z} \,\}$ and $F = \{\, q \in Q \mid \varrho[q] \in \mathbb{Z} \,\}$. We denote $t_1 = [\![I, \mu(u_1), \alpha(\mu(u_2)) \cdot F]\!] = \left(\begin{smallmatrix} I & & P_1 \\ R_0 & M_1 & R_1 \end{smallmatrix}\right)$ and $t_2 = [\![I \cdot \alpha(\mu(u_1)), \mu(u_2), F]\!] = \left(\begin{smallmatrix} P_1 & & P_2 \\ R_1 & M_2 & F \end{smallmatrix}\right)$. Then, $t_1t_2 = \left(\begin{smallmatrix} I & & P_2 \\ R_0 & M_1M_2 & F \end{smallmatrix}\right)$, and moreover, $|\mathcal{A}|(u_1u_2)$ is the least entry in M_1M_2, i.e., $|\mathcal{A}|(u_1u_2) = \mathsf{min}(M_1M_2)$.

Let $p_0 \in I \cap R_0$, let $p_1 \in P_1 \cap R_1$, and let $p_2 \in P_2 \cap F$. Assume $(M_1, M_2)[p_0, p_1, p_2] \in \mathbb{Z}$.

Moreover, assume that $M_1[p_0, p_1] \geq \mathsf{min}(M_1) + 2Y$, (the Y from Proposition 5.1) but nevertheless $(M_1, M_2)[p_0, p_1, p_2] = \mathsf{min}(M_1M_2)$. Intuitively, the path along p_0, p_1, p_2 has after reading u_1 from p_0 to p_1 a very large weight (in comparison to the path which has a weight of $\mathsf{min}(M_1)$), but nevertheless, by reading u_2 from p_1 to p_2 the weight of the path becomes smaller and smaller and finally the path has a weight of $\mathsf{min}(M_1M_2)$, i.e., it is the path with the least weight.

Let $q_0 \in I \cap R_0$, let $q_1 \in P_1 \cap R_1$, and let $q_2 \in P_2 \cap F$. Assume $(M_1, M_2)[q_0, q_1, q_2] \in \mathbb{Z}$.

We have $(M_1, M_2)[q_0, q_1, q_2] \geq \mathsf{min}(M_1M_2) = (M_1, M_2)[p_0, p_1, p_2]$. Hence, we have $M_1[q_0, q_1] \geq M_1[p_0, p_1] - Y$ or $M_2[q_1, q_2] \geq M_2[p_1, p_2] + Y \geq \mathsf{min}(M_2) + Y$. However, $M_1[q_0, q_1] \geq M_1[p_0, p_1] - Y$ implies $M_1[q_0, q_1] \geq \mathsf{min}(M_1) + Y$ (by the above assumption on $M_1[p_0, p_1]$). Consequently, we have $M_1[q_0, q_1] \geq \mathsf{min}(M_1) + Y$ or $M_2[q_1, q_2] \geq \mathsf{min}(M_2) + Y$.

Now, let t'_1 and t'_2 be the matrices which exist by Proposition 5.1. By (A2), we have $M'_1[q_0, q_1] = \omega$ or $M'_2[q_1, q_2] = \omega$ whereas M'_1 resp. M'_2 are the matrices in t'_1 resp. t'_2.

Since this argumentation holds for every q_0, q_1, q_2 (in particular for p_0, p_1, p_2) every entry of $t'_1t'_2$ is ω or ∞, i.e., $t'_1t'_2$ shows that (1) in Theorem 3.4 is satisfied, which is a contradiction.

Consequently, the above assumed p_0, p_1, p_2 cannot exist.

Let $w \in \Sigma^*$, and let π be an accepting path. Assume the weight of π is $|\mathcal{A}|(w)$. By the above observation, π can intermediately not have a much larger (i.e. $2Y$ larger) weight than another accepting path.

Acknowledgement

The authors thank anonymous referees for their useful remarks.

References

[1] S. Gaubert. On the Burnside problem for semigroups of matrices in the $(\mathsf{max}, +)$ algebra. *Semigroup Forum*, 52:271–292, 1996.
[2] S. Gaubert and J. Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE Trans. Aut. Cont.*, 44(4):683–698, 1998.
[3] G. Grahne and A. Thomo. Approximate reasoning in semi-structured databases. In M. Lenzerini et al., ed., *KRDB2001 Proceedings*, vol. 45 of *CEUR Workshop Proceedings*, 2001.
[4] K. Hashiguchi. A decision procedure for the order of regular events. *Theor. Comp. Sc.*, 8:69–72, 1979.
[5] K. Hashiguchi. Limitedness theorem on finite automata with distance functions. *Journal of Computer and System Sciences*, 24:233–244, 1982.
[6] K. Hashiguchi. Algorithms for determining relative star height and star height. *Information and Computation*, 78:124–169, 1988.
[7] K. Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theoretical Computer Science*, 72(1):27–38, 1990.
[8] K. Hashiguchi, K. Ishiguro, and S. Jimbo. Decidability of the equivalence problem for finitely ambiguous finance automata. *International Journal of Algebra and Computation*, 12(3):445–461, 2002.
[9] N. Haubold. The similarity and equivalence problem for finitely ambiguous automata over the tropical semiring. Master's thesis, Technische Universität Dresden, Institut für Algebra, 2006.
[10] J. Hromkovič, J. Karhumäki, H. Klauck, G. Schnitger, and S. Seibert. Communication complexity method for measuring nondeterminism in finite automata. *Inf. and Comp.*, 172(2):202–217, 2002.
[11] O. Ibarra and B. Ravikumar. On sparseness, ambiguity and other decision problems for acceptors and transducers. In B. Monien, G. Vidal-Naquet, *STACS'86*, LNCS 210, p. 171–179. Springer-Verlag, 1986.
[12] D. Kirsten. Distance desert automata and the star height problem. *R.A.I.R.O. - Informatique Théorique et Applications, special issue of selected best papers from FoSSaCS 2004*, 39(3):455–509, 2005.
[13] D. Kirsten. Distance desert automata and star height substitutions. Habilitationsschrift, Universität Leipzig, Fakultät für Mathematik und Informatik, 2006.
[14] D. Kirsten. A Burnside approach to the termination of Mohri's algorithm for polynomially ambiguous min-plus-automata. *R.A.I.R.O. - ITA, special issue on "JM'06"*, 42:553–581, 2008.
[15] I. Klimann, S. Lombardy, J. Mairesse, and C. Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.
[16] D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–425, 1994.
[17] D. Krob. Some consequences of a Fatou property of the tropical semiring. *Journal of Pure and Applied Algebra*, 93:231–249, 1994.
[18] H. Leung. The topological approach to the limitedness problem on distance automata. In J. Gunawardena, editor, *Idempotency*, pages 88–111. Cambridge University Press, 1998.
[19] S. Lombardy and J. Sakarovitch. Sequential? *Theoretical Computer Science*, 356:224–244, 2006.
[20] Y. Métivier and G. Richomme. New results on the star problem in trace monoids. *Information and Computation*, 119(2):240–251, 1995.
[21] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311, 1997.
[22] H. Seidl and A. Weber. On the degree of ambiguity of finite automata. *Th. C. Sc.*, 88:325–349, 1991.
[23] I. Simon. Recognizable sets with multiplicities in the tropical semiring. In M. P. Chytil et al., editors, *MFCS'88 Proceedings*, volume 324 of LNCS, pages 107–120. Springer-Verlag, Berlin, 1988.
[24] I. Simon. On semigroups of matrices over the tropical semiring. *R.A.I.R.O. - Informatique Théorique et Applications*, 28:277–294, 1994.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 601–612
www.stacs-conf.org

POLYNOMIAL KERNELIZATIONS FOR MIN $F^+\Pi_1$ AND MAX NP

STEFAN KRATSCH [1]

[1] Max-Planck-Institut für Informatik, Campus E 1 4, 66123 Saarbrücken, Germany
E-mail address: skratsch@mpi-inf.mpg.de
URL: http://www.mpi-inf.mpg.de/~skratsch

Abstract. The relation of constant-factor approximability to fixed-parameter tractability and kernelization is a long-standing open question. We prove that two large classes of constant-factor approximable problems, namely MIN $F^+\Pi_1$ and MAX NP, including the well-known subclass MAX SNP, admit polynomial kernelizations for their natural decision versions. This extends results of Cai and Chen (JCSS 1997), stating that the standard parameterizations of problems in MAX SNP and MIN $F^+\Pi_1$ are fixed-parameter tractable, and complements recent research on problems that do not admit polynomial kernelizations (Bodlaender et al. ICALP 2008).

1. Introduction

The class APX consists of all NP optimization problems that are approximable to within a constant factor of the optimum. It is known that the decision versions of most APX-problems are fixed-parameter tractable or even admit efficient preprocessing in the form of a polynomial kernelization. How strong is the relation between constant-factor approximability and polynomial kernelizability? Is there a property inherent to most APX-problems that explains this relation? What is the nature of APX-problems that do not admit a polynomial kernelization, such as Bin Packing for example?

Since many prominent APX-problems are complete under approximation preserving reductions and do not admit arbitrarily small approximation ratios, studying their parameterized complexity is a natural approach to obtain better results (recently Cai and Huang presented fixed-parameter approximation schemes for MAX SNP [7]). In conjunction with recent work on problems without polynomial kernelizations, positive answers to the questions may provide evidence against APX-membership for some problems (e.g. Treewidth).

Our work: We prove that the standard parameterizations of problems in two large classes of constant-factor approximable problems, namely MIN $F^+\Pi_1$ and MAX NP, admit polynomial kernelizations. This extends results of Cai and Chen [6] who showed that the standard parameterizations of all problems in MIN $F^+\Pi_1$ and MAX SNP (a subclass of MAX NP) are fixed-parameter tractable.[1] Interestingly perhaps, both our results rely on the Sunflower Lemma due to Erdős and Rado [10].

1998 ACM Subject Classification: F.2.2.
Key words and phrases: parameterized complexity, kernelization, approximation algorithms.
[1]The existence of a kernelization, not necessarily polynomial, is equivalent to fixed-parameter tractability.

	Approximation ratio	Kernel size
Minimum Vertex Cover	2 [15]	$O(k)$ [8]
Feedback Vertex Set	2 [3]	$O(k^3)$ [4]
Minimum Fill-In	$O(\text{opt})$ [19]	$O(k^2)$ [19]
Treewidth	$O(\sqrt{\log \text{opt}})$ [12]	not poly[2] [5]

Table 1: Approximation ratio and size of problem kernels for some optimization problems.

Related work: Recently Bodlaender et al. [5] presented the first negative results concerning the existence of polynomial kernelizations for some natural fixed-parameter tractable problems. Using the notion of a *distillation algorithm* and results due to Fortnow and Santhanam [14], they were able to show that the existence of polynomial kernelizations for so-called *compositional* parameterized problems implies a collapse of the polynomial hierarchy to the third level. These are seminal results presenting the first super-linear lower bounds for kernelization and relating a statement from parameterized complexity to a hypothesis from classical complexity theory.

In Table 1 we summarize approximability and kernelization results for some well-known problems.

MIN $F^+\Pi_1$ and MAX NP: Two decades ago Papadimitriou and Yannakakis [23] initiated the syntactic study of optimization problems to extend the understanding of approximability. They introduced the classes MAX NP and MAX SNP as natural variants of NP based on Fagin's [11] syntactic characterization of NP. Essentially problems are in MAX NP or MAX SNP if their optimum value can be expressed as the maximum number of tuples for which some existential, respectively quantifier-free, first-order formula holds. They showed that every problem in these two classes is approximable to within a constant factor of the optimum. Arora et al. complemented this by proving that no MAX SNP-complete problem has a polynomial-time approximation scheme, unless P=NP [2]. Contained in MAX SNP there are some well-known maximization problems, such as Max Cut, Max q-Sat, and Independent Set on graphs of bounded degree. Its superclass MAX NP also contains Max Sat amongst others.

Kolaitis and Thakur generalized the approach of examining the logical definability of optimization problems and defined further classes of minimization and maximization problems [17, 18]. Amongst others they introduced the class MIN $F^+\Pi_1$ of problems whose optimum can be expressed as the minimum weight of an assignment (i.e. number of ones) that satisfies a certain universal first-order formula. They proved that every problem in MIN $F^+\Pi_1$ is approximable to within a constant factor of the optimum. In MIN $F^+\Pi_1$ there are problems like Vertex Cover, d-Hitting Set, and Triangle Edge Deletion.

Section 2 covers the definitions of the classes MIN $F^+\Pi_1$ and MAX NP, as well as the necessary details from parameterized complexity. In Sections 3 and 4 we present polynomial kernelizations for the standard parameterizations of problems in MIN $F^+\Pi_1$ and MAX NP respectively. Section 5 summarizes our results and poses some open problems.

[2]Treewidth does not admit a polynomial kernelization unless there is a distillation algorithm for all coNP complete problems [5]. Though unlikely, this is not known to imply a collapse of the polynomial hierarchy.

2. Preliminaries

Logic and complexity classes: A (relational) vocabulary is a set σ of relation symbols, each having some fixed integer as its arity. Atomic formulas over σ are of the form $R(z_1, \ldots, z_t)$ where R is a t-ary relation symbol from σ and the z_i are variables. The set of quantifier-free (relational) formulas over σ is the closure of the set of all atomic formulas under negation, conjunction, and disjunction.

Definition 2.1 (MIN $\mathrm{F}^+\Pi_1$, MAX NP)**.** A *finite structure of type* $(r_1, \ldots, r_t)$ is a tuple $\mathcal{A} = (A, R_1, \ldots, R_t)$ where A is a finite set and each R_i is an r_i-ary relation over A.

Let $\mathcal{Q}$ be an optimization problem on finite structures of type $(r_1, \ldots, r_t)$. Let $R_1, \ldots, R_t$ be relation symbols of arity $r_1, \ldots, r_t$.
(a) The problem $\mathcal{Q}$ is contained in the class MIN $\mathrm{F}^+\Pi_1$ if its optimum on finite structures $\mathcal{A}$ of type $(r_1, \ldots, r_t)$ can be expressed as

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \min_{S}\{|S| : (\mathcal{A}, S) \models (\forall \mathbf{x} \in A^{c_x}) : \psi(\mathbf{x}, S)\},$$

where S is a single relation symbol and $\psi(\mathbf{x}, S)$ is a quantifier-free formula in conjunctive normal form over the vocabulary $\{R_1, \ldots, R_t, S\}$ on variables $\{x_1, \ldots, x_{c_x}\}$. Furthermore, $\psi(\mathbf{x}, S)$ is positive in S, i.e. S does not occur negated in $\psi(\mathbf{x}, S)$.
(b) The problem $\mathcal{Q}$ is contained in the class MAX NP if its optimum on finite structures $\mathcal{A}$ of type $(r_1, \ldots, r_t)$ can be expressed as

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \max_{\mathcal{S}} \left|\{\mathbf{x} \in A^{c_x} : (\mathcal{A}, \mathcal{S}) \models (\exists \mathbf{y} \in A^{c_y}) : \psi(\mathbf{x}, \mathbf{y}, \mathcal{S})\}\right|,$$

where $\mathcal{S} = (S_1, \ldots, S_u)$ is a tuple of s_i-ary relation symbols S_i and $\psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$ is a quantifier-free formula in disjunctive normal form over the vocabulary $\{R_1, \ldots, R_t, S_1, \ldots, S_u\}$ on variables $\{x_1, \ldots, x_{c_x}, y_1, \ldots, y_{c_y}\}$.

Remark 2.2. The definition of MAX SNP is similar to that of MAX NP but without the existential quantification of $\mathbf{y}$, i.e $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \max_{\mathcal{S}} |\{\mathbf{x} : (\mathcal{A}, \mathcal{S}) \models \psi(\mathbf{x}, \mathcal{S})\}|$.

Example 2.3 (Minimum Vertex Cover)**.** Let $G = (V, E)$ be a finite structure of type (2) that represents a graph by a set V of vertices and a binary relation E over V as its edges. The optimum of Minimum Vertex Cover on structures G can be expressed as:

$$\mathrm{opt}_{VC}(G) = \min_{S \subseteq V}\{|S| : (G, S) \models (\forall (u, v) \in V^2) : (\neg E(u, v) \vee S(u) \vee S(v))\}.$$

This implies that Minimum Vertex Cover is contained in MIN $\mathrm{F}^+\Pi_1$.

Example 2.4 (Maximum Satisfiability)**.** Formulas in conjunctive normal form can be represented by finite structures $\mathcal{F} = (F, P, N)$ of type $(2, 2)$: Let F be the set of all clauses and variables, and let P and N be binary relations over F. Let $P(x, c)$ be true if and only if x is a literal of the clause c and let $N(x, c)$ be true if and only if $\neg x$ is a literal of the clause c. The optimum of Max Sat on structures $\mathcal{F}$ can be expressed as:

$$\mathrm{opt}_{MS}(\mathcal{F}) = \max_{T \subseteq F} |\{c \in F : (\mathcal{F}, T) \models (\exists x \in F) : (P(x, c) \wedge T(x)) \vee (N(x, c) \wedge \neg T(x))\}|.$$

Thus Max Sat is contained in MAX NP.

For a detailed introduction to MIN $\mathrm{F}^+\Pi_1$, MAX NP, and MAX SNP we refer the reader to [17, 18, 23]. An introduction to logic and complexity can be found in [22].

Parameterized complexity: The field of parameterized complexity, pioneered by Downey and Fellows, is a two-dimensional approach of coping with combinatorially hard problems.

Parameterized problems come with a parameterization that maps input instances to a parameter value. The time complexity of algorithms is measured with respect to the input size and the parameter. In the following we give the necessary formal definitions, namely fixed-parameter tractability, standard parameterizations, and kernelization.

Definition 2.5 (Fixed-parameter tractability)**.** A *parameterization* of Σ^* is a polynomial-time computable mapping $\kappa : \Sigma^* \to \mathbb{N}$. A *parameterized problem* over an alphabet Σ is a pair $(\mathcal{Q}, \kappa)$ consisting of a set $\mathcal{Q} \subseteq \Sigma^*$ and a parameterization κ of Σ^*.

A parameterized problem $(\mathcal{Q}, \kappa)$ is *fixed-parameter tractable* if there exists an algorithm $\mathbb{A}$, a polynomial p, and a computable function $f : \mathbb{N} \to \mathbb{N}$ such that $\mathbb{A}$ decides $x \in \mathcal{Q}$ in time $f(\kappa(x)) \cdot p(|x|)$. FPT is the class of all fixed-parameter tractable problems.

Definition 2.6 (Standard parameterization)**.** Let $\mathcal{Q}$ be a maximization (minimization) problem. The *standard parameterization* of $\mathcal{Q}$ is $p\text{-}\mathcal{Q} = (d\text{-}\mathcal{Q}, \kappa)$ where $\kappa : (\mathcal{A}, k) \mapsto k$ and $d\text{-}\mathcal{Q}$ is the language of all tuples $(\mathcal{A}, k)$ such that $\text{opt}_{\mathcal{Q}}(\mathcal{A}) \geq k$ ($\text{opt}_{\mathcal{Q}}(\mathcal{A}) \leq k$).

Basically $d\text{-}\mathcal{Q}$ is the decision version of $\mathcal{Q}$, asking whether the optimum is at least k (respectively at most k). The standard parameterization of $\mathcal{Q}$ is $d\text{-}\mathcal{Q}$ parameterized by k.

Definition 2.7 (Kernelization)**.** Let $(\mathcal{Q}, \kappa)$ be a parameterized problem over Σ. A polynomial-time computable function $K : \Sigma^* \to \Sigma^*$ is a *kernelization* of $(\mathcal{Q}, \kappa)$ if there is a computable function $h : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \Sigma^*$ we have

$$(x \in \mathcal{Q} \Leftrightarrow K(x) \in \mathcal{Q}) \text{ and } |K(x)| \leq h(\kappa(x)).$$

We call h the *size* of the *problem kernel* $K(x)$. The kernelization K is *polynomial* if h is a polynomial. We say that $(\mathcal{Q}, \kappa)$ *admits a (polynomial) kernelization* if there exists a (polynomial) kernelization of $(\mathcal{Q}, \kappa)$.

Essentially, a kernelization is a polynomial-time data reduction that comes with a guaranteed upper bound on the size of the resulting instance in terms of the parameter.

For an introduction to parameterized complexity we refer the reader to [9, 13, 20].

Hypergraphs and sunflowers: We assume the reader to be familiar with the basic graph notation. A *hypergraph* is a tuple $\mathcal{H} = (V, E)$ consisting of a finite set V, its vertices, and a family E of subsets of V, its edges. A hypergraph has *dimension* d if each edge has cardinality at most d. A hypergraph is *d-uniform* if each edge has cardinality exactly d.

Definition 2.8 (Sunflower)**.** Let $\mathcal{H}$ be a hypergraph. A *sunflower* of cardinality r is a set $F = \{f_1, \ldots, f_r\}$ of edges of $\mathcal{H}$ such that every pair has the same intersection C, i.e. for all $1 \leq i < j \leq r$: $f_i \cap f_j = C$. The set C is called the *core* of the sunflower.

Note that any family of pairwise disjoint sets is a sunflower with core $C = \emptyset$.

Lemma 2.9 (Sunflower Lemma [10])**.** *Let $k, d \in \mathbb{N}$ and let $\mathcal{H}$ be a d-uniform hypergraph with more than $(k-1)^d \cdot d!$ edges. Then there is a sunflower of cardinality k in $\mathcal{H}$. For every fixed d there is an algorithm that computes such a sunflower in time polynomial in $|E(\mathcal{H})|$.*

Corollary 2.10 (Sunflower Corollary)**.** *The same holds for d-dimensional hypergraphs with more than $(k-1)^d \cdot d! \cdot d$ edges.*

Proof. For some $d' \in \{1, \ldots, d\}$, $\mathcal{H}$ has more than $(k-1)^d \cdot d! \geq (k-1)^{d'} \cdot d'!$ edges of cardinality d'. Let $\mathcal{H}_{d'}$ be the d'-uniform subgraph induced by the edges of cardinality d'. We apply the Sunflower Lemma on $\mathcal{H}_{d'}$ and obtain a sunflower F of cardinality k in time polynomial in $|E(\mathcal{H}_{d'})| \leq |E(\mathcal{H})|$. Clearly F is also a sunflower of $\mathcal{H}$. ∎

3. Polynomial kernelization for MIN $\mathrm{F}^+\Pi_1$

We will prove that the standard parameterization of any problem in MIN $\mathrm{F}^+\Pi_1$ admits a polynomial kernelization. The class MIN $\mathrm{F}^+\Pi_1$ was introduced by Kolaitis and Thakur in a framework of syntactically defined classes of optimization problems [17]. In a follow-up paper they showed that every problem in MIN $\mathrm{F}^+\Pi_1$ is constant-factor approximable [18].

Throughout the section let $\mathcal{Q} \in$ MIN $\mathrm{F}^+\Pi_1$ be an optimization problem on finite structures of type $(r_1, \ldots, r_t)$. Let $R_1, \ldots, R_t$ be relation symbols of arity $r_1, \ldots, r_t$ and let S be a relation symbol of arity c_S. Furthermore, let $\psi(\mathbf{x}, S)$ be a quantifier-free formula in conjunctive normal form over the vocabulary $\{R_1, \ldots, R_t, S\}$ on variables $\{x_1, \ldots, x_{c_x}\}$ that is positive in S such that

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \min_{S \subseteq A^{c_S}} \{|S| : (\mathcal{A}, S) \models (\forall \mathbf{x} \in A^{c_x}) : \psi(\mathbf{x}, S)\}.$$

Let s be the maximum number of occurrences of S in any clause of $\psi(\mathbf{x}, S)$. The standard parameterization p-$\mathcal{Q}$ of $\mathcal{Q}$ is the following problem:

Input: A finite structure $\mathcal{A}$ of type $(r_1, \ldots, r_t)$ and an integer k.
Parameter: k.
Task: Decide whether $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) \leq k$.

We will see that, given an instance $(\mathcal{A}, k)$, deciding whether $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) \leq k$ is equivalent to deciding an instance of s-Hitting Set.[3] Our kernelization will therefore make use of existing kernelization results for s-Hitting Set. The parameterized version of s-Hitting Set is defined as follows:

Input: A hypergraph $\mathcal{H} = (V, E)$ of dimension s and an integer k.
Parameter: k.
Task: Decide whether $\mathcal{H}$ has a hitting set of size at most k, i.e. $S \subseteq V$, $|S| \leq k$, such that S has a nonempty intersection with every edge of $\mathcal{H}$.

We consider the formula $\psi(\mathbf{x}, S)$ and a fixed instance $(\mathcal{A}, k)$, with $\mathcal{A} = (A, R_1, \ldots, R_t)$. For every tuple $\mathbf{x} \in A^{c_x}$ we can evaluate all literals of the form $R_i(\mathbf{z})$ and $\neg R_i(\mathbf{z})$ for some $\mathbf{z} \in \{x_1, \ldots, x_{c_x}\}^{r_i}$. By checking whether $\mathbf{z} \in R_i$, we obtain 1 (true) or 0 (false) for each literal. Then we delete all occurrences of 0 from the clauses and delete all clauses that contain a 1. For each $\mathbf{x}$, we obtain an equivalent formula that we denote with $\psi_{\mathbf{x}}(S)$. Each $\psi_{\mathbf{x}}(S)$ is in conjunctive normal form on literals $S(\mathbf{z})$ for some $\mathbf{z} \in \{x_1, \ldots, x_{c_x}\}^{c_S}$ (no literals of the form $\neg S(\mathbf{z})$ since $\psi(\mathbf{x}, S)$ is positive in S).

Remark 3.1. For all $\mathbf{x} \in A^{c_x}$ and $S \subseteq A^{c_S}$ it holds that $(\mathcal{A}, S) \models \psi(\mathbf{x}, S)$ if and only if $(\mathcal{A}, S) \models \psi_{\mathbf{x}}(S)$. Moreover, we can compute all formulas $\psi_{\mathbf{x}}(S)$ for $\mathbf{x} \in A^{c_x}$ in polynomial time, since c_x and the length of $\psi(\mathbf{x}, S)$ are constants independent of $\mathcal{A}$.

Deriving a formula $\psi_{\mathbf{x}}(S)$ can yield empty clauses. This happens when all literals $R_i(\cdot)$, $\neg R_i(\cdot)$ in a clause are evaluated to 0 and there are no literals $S(\cdot)$. In that case, no assignment S can satisfy the formula $\psi_{\mathbf{x}}(S)$, or equivalently $\psi(\mathbf{x}, S)$. Thus $(\mathcal{A}, k)$ is a no-instance. Note that clauses of $\psi_{\mathbf{x}}(S)$ cannot contain contradicting literals since $\psi(\mathbf{x}, S)$ is positive in S.

Remark 3.2. From now on, we assume that all clauses of the formulas $\psi_{\mathbf{x}}(S)$ are nonempty.

We define a mapping Φ from finite structures $\mathcal{A}$ to hypergraphs $\mathcal{H}$. Then we show that equivalent s-Hitting Set instances can be obtained in this way.

[3] In literature the problem is often called d-Hitting Set but we will need $d = s$.

Definition 3.3. Let $\mathcal{A}$ be an instance of $\mathcal{Q}$. We define $\Phi(\mathcal{A}) := \mathcal{H}$ with $\mathcal{H} = (V, E)$. We let E be the family of all sets $e = \{\mathbf{z}_1, \ldots, \mathbf{z}_p\}$ such that $(S(\mathbf{z}_1) \vee \cdots \vee S(\mathbf{z}_p))$ is a clause of a $\psi_{\mathbf{x}}(S)$ for some $\mathbf{x} \in A^{c_x}$. We let V be the union of all sets $e \in E$.

Remark 3.4. The hypergraphs $\mathcal{H}$ obtained from the mapping Φ have dimension s since each $\psi_{\mathbf{x}}(S)$ has at most s literals per clause. It follows from Remark 3.1 that $\Phi(\mathcal{A})$ can be computed in polynomial time.

The following lemma establishes that $(\mathcal{A}, k)$ and $(\mathcal{H}, k) = (\Phi(\mathcal{A}), k)$ are equivalent in the sense that $(\mathcal{A}, k) \in p\text{-}\mathcal{Q}$ if and only if $(\mathcal{H}, k) \in s$-Hitting Set.

Lemma 3.5. *Let $\mathcal{A} = (A, R_1, \ldots, R_t)$ be an instance of $\mathcal{Q}$ then for all $S \subseteq A^{c_S}$:*

$$(\mathcal{A}, S) \models (\forall \mathbf{x}) : \psi(\mathbf{x}, S) \text{ if and only if } S \text{ is a hitting set for } \mathcal{H} = \Phi(\mathcal{A}).$$

Proof. Let $\mathcal{H} = \Phi(\mathcal{A}) = (V, E)$ and let $S \subseteq A^{c_S}$:

$$\begin{aligned}
& (\mathcal{A}, S) \models (\forall \mathbf{x} \in A^{c_x}) : \psi(\mathbf{x}, S) \\
\Leftrightarrow\ & (\mathcal{A}, S) \models (\forall \mathbf{x} \in A^{c_x}) : \psi_{\mathbf{x}}(S) \\
\Leftrightarrow\ & (\forall \mathbf{x} \in A^{c_x}) : \text{each clause of } \psi_{\mathbf{x}}(S) \text{ has a literal } S(\mathbf{z}) \text{ for which } \mathbf{z} \in S \\
\Leftrightarrow\ & S \text{ has a nonempty intersection with every set } e \in E \\
\Leftrightarrow\ & S \text{ is a hitting set for } (V, E).
\end{aligned}$$

■

Our kernelization will consist of the following steps:

(1) Map the given instance $(\mathcal{A}, k)$ for p-$\mathcal{Q}$ to an equivalent instance $(\mathcal{H}, k) = (\Phi(\mathcal{A}), k)$ for s-Hitting Set according to Definition 3.3 and Lemma 3.5.
(2) Use a polynomial kernelization for s-Hitting Set on $(\mathcal{H}, k)$ to obtain an equivalent instance $(\mathcal{H}', k)$ with size polynomial in k.
(3) Use $(\mathcal{H}', k)$ to derive an equivalent instance $(\mathcal{A}', k)$ of p-$\mathcal{Q}$. That way we will be able to conclude that $(\mathcal{A}', k)$ is equivalent to $(\mathcal{H}, k)$ and hence also to $(\mathcal{A}, k)$.

There exist different kernelizations for s-Hitting Set: one by Flum and Grohe [13] based on the Sunflower Lemma due to Erdős and Rado [10], one by Nishimura et al. [21] via a generalization of the Nemhauser-Trotter kernelization for Vertex Cover, and a recent one by Abu-Khzam [1] based on crown decompositions. For our purposes of deriving an equivalent instance for p-$\mathcal{Q}$, these kernelizations have the drawback of shrinking sets during the reduction. This is not possible for our approach since we would need to change the formula $\psi(\mathbf{x}, S)$ to shrink the clauses. We prefer to modify Flum and Grohe's kernelization such that it uses only edge deletions.

Theorem 3.6. *There exists a polynomial kernelization of s-Hitting Set that, given an instance $(\mathcal{H}, k)$, computes an instance $(\mathcal{H}^*, k)$ such that $E(\mathcal{H}^*) \subseteq E(\mathcal{H})$, $\mathcal{H}^*$ has $O(k^s)$ edges, and the size of $(\mathcal{H}^*, k)$ is $O(k^s)$ as well.*

Proof. Let $(\mathcal{H}, k)$ be an instance of s-Hitting Set, with $\mathcal{H} = (V, E)$. If $\mathcal{H}$ contains a sunflower $F = \{f_1, \ldots, f_{k+1}\}$ of cardinality $k + 1$ then every hitting set of $\mathcal{H}$ must have a nonempty intersection with the core C of F or with the $k+1$ disjoint sets $f_1 \setminus C, \ldots, f_{k+1} \setminus C$. Thus every hitting set of at most k elements must have a nonempty intersection with C.

Now consider a sunflower $F = \{f_1, \ldots, f_{k+1}, f_{k+2}\}$ of cardinality $k+2$ in $\mathcal{H}$ and let $\mathcal{H}' = (V, E \setminus \{f_{k+2}\})$. We show that the instances $(\mathcal{H}, k)$ and $(\mathcal{H}', k)$ are equivalent. Clearly every hitting set for $\mathcal{H}$ is also a hitting set for $\mathcal{H}'$ since $E(\mathcal{H}') \subseteq E(\mathcal{H})$. Let $S \subseteq V$ be a hitting set of size at most k for $\mathcal{H}'$. Since $F \setminus \{f_{k+2}\}$ is a sunflower of cardinality $k+1$ in $\mathcal{H}'$, it follows that S has a nonempty intersection with its core C. Hence S has a nonempty intersection

with $f_{k+2} \supseteq C$ too. Thus S is a hitting set of size at most k for $\mathcal{H}$, implying that $(\mathcal{H}, k)$ and $(\mathcal{H}', k)$ are equivalent.

We start with $\mathcal{H}^* = \mathcal{H}$ and repeat the following step while $\mathcal{H}^*$ has more than $(k+1)^s \cdot s! \cdot s$ edges. By the Sunflower Corollary we obtain a sunflower of cardinality $k+2$ in $\mathcal{H}^*$ in time polynomial in $|E(\mathcal{H}^*)|$. We delete an edge of the detected sunflower from the edge set of $\mathcal{H}^*$ (thereby reducing the cardinality of the sunflower to $k+1$). Thus, by the argument from the previous paragraph, we maintain that $(\mathcal{H}, k)$ and $(\mathcal{H}^*, k)$ are equivalent.

Furthermore $E(\mathcal{H}^*) \subseteq E(\mathcal{H})$ and $\mathcal{H}^*$ has no more than $(k+1)^s \cdot s! \cdot s \in O(k^s)$ edges. Since we delete an edge of $\mathcal{H}^*$ in each step, there are $O(|E(\mathcal{H})|)$ steps, and the total time is polynomial in $|E(\mathcal{H})|$. Deleting all isolated vertices from $\mathcal{H}^*$ yields a size of $O(s \cdot k^s) = O(k^s)$ since each edge contains at most s vertices. ∎

The following lemma proves that every s-Hitting Set instance that is "sandwiched" between two equivalent instances must be equivalent to both.

Lemma 3.7. *Let $(\mathcal{H}, k)$ be an instance of s-Hitting Set and let $(\mathcal{H}^*, k)$ be an equivalent instance with $E(\mathcal{H}^*) \subseteq E(\mathcal{H})$. Then for any $\mathcal{H}'$ with $E(\mathcal{H}^*) \subseteq E(\mathcal{H}') \subseteq E(\mathcal{H})$ the instance $(\mathcal{H}', k)$ is equivalent to $(\mathcal{H}, k)$ and $(\mathcal{H}^*, k)$.*

Proof. Observe that hitting sets for $\mathcal{H}$ can be projected to hitting sets for $\mathcal{H}'$ (i.e. restricted to the vertex set of $\mathcal{H}'$) since $E(\mathcal{H}') \subseteq E(\mathcal{H})$. Thus if $(\mathcal{H}, k)$ is a yes-instance then $(\mathcal{H}', k)$ is a yes-instance too. The same argument holds for $(\mathcal{H}', k)$ and $(\mathcal{H}^*, k)$. Together with the fact that $(\mathcal{H}, k)$ and $(\mathcal{H}^*, k)$ are equivalent, this proves the lemma. ∎

Now we are well equipped to prove that p-$\mathcal{Q}$ admits a polynomial kernelization.

Theorem 3.8. *Let $\mathcal{Q} \in$ MIN $\mathrm{F}^+\Pi_1$. The standard parameterization p-$\mathcal{Q}$ of $\mathcal{Q}$ admits a polynomial kernelization.*

Proof. Let $(\mathcal{A}, k)$ be an instance of p-$\mathcal{Q}$. By Lemma 3.5 we have that $(\mathcal{A}, k)$ is a yes-instance of p-$\mathcal{Q}$ if and only if $(\mathcal{H}, k) = (\Phi(\mathcal{A}), k))$ is a yes-instance of s-Hitting Set. We apply the kernelization from Theorem 3.6 to $(\mathcal{H}, k)$ and obtain an equivalent s-Hitting Set instance $(\mathcal{H}^*, k)$ such that $E(\mathcal{H}^*) \subseteq E(\mathcal{H})$ and $\mathcal{H}^*$ has $O(k^s)$ edges.

Recall that every edge of $\mathcal{H}$, say $\{\mathbf{z}_1, \ldots, \mathbf{z}_p\}$, corresponds to a clause $(S(\mathbf{z}_1) \vee \cdots \vee S(\mathbf{z}_p))$ of $\psi_{\mathbf{x}}(S)$ for some $\mathbf{x} \in A^{c_x}$. Thus for each edge $e \in E(\mathcal{H}^*) \subseteq E(\mathcal{H})$ we can select a tuple $\mathbf{x}_e$ such that e corresponds to a clause of $\psi_{\mathbf{x}_e}(S)$. Let X be the set of the selected tuples $\mathbf{x}_e$ for all edges $e \in E(\mathcal{H}^*)$. Let $A' \subseteq A$ be the set of all components of tuples $\mathbf{x}_e \in X$, ensuring that $X \subseteq A'^{c_x}$. Let R'_i be the restriction of R_i to A' and let $\mathcal{A}' = (A', R'_1, \ldots, R'_t)$.

Let $(\mathcal{H}', k) = (\Phi(\mathcal{A}'), k)$. By definition of Φ and by construction of $\mathcal{H}'$ we know that $E(\mathcal{H}^*) \subseteq E(\mathcal{H}') \subseteq E(\mathcal{H})$ since $X \subseteq A'^{c_x}$ and $A' \subseteq A$. Thus, by Lemma 3.7, we have that $(\mathcal{H}', k)$ is equivalent to $(\mathcal{H}, k)$. Furthermore, by Lemma 3.5, $(\mathcal{H}', k)$ is a yes-instance of s-Hitting Set if and only if $(\mathcal{A}', k)$ is a yes-instance of p-$\mathcal{Q}$. Thus $(\mathcal{A}', k)$ and $(\mathcal{A}, k)$ are equivalent instances of p-$\mathcal{Q}$.

We conclude the proof by giving an upper bound on the size of $(\mathcal{A}', k)$ that is polynomial in k. The set X contains at most $|E(\mathcal{H}^*)| \in O(k^s)$ tuples. These tuples have no more than $c_x \cdot |E(\mathcal{H}^*)|$ different components. Hence the size of A' is $O(c_x \cdot k^s) = O(k^s)$. Thus the size of $(\mathcal{A}', k)$ is $O(k^{sm})$, where m is the largest arity of a relation R_i. The values c_x, s, and m are constants that are independent of the input $(\mathcal{A}, k)$. Thus $(\mathcal{A}', k)$ is an instance equivalent to $(\mathcal{A}, k)$ with size polynomial in k. ∎

4. Polynomial kernelization for MAX NP

We prove that the standard parameterization of any problem in MAX NP admits a polynomial kernelization. The class MAX NP was introduced by Papadimitriou and Yannakakis in [23]. They showed that every problem in MAX NP is constant-factor approximable.

Throughout the section let $\mathcal{Q} \in$ MAX NP be an optimization problem on finite structures of type $(r_1, \ldots, r_t)$. Let $R_1, \ldots, R_t$ be relation symbols of arity $r_1, \ldots, r_t$ and let $\mathcal{S} = (S_1, \ldots, S_u)$ be a tuple of relation symbols of arity $s_1, \ldots, s_u$. Let $\psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$ be a formula in disjunctive normal form over the vocabulary $\{R_1, \ldots, R_t, S_1, \ldots, S_u\}$ on variables $\{x_1, \ldots, x_{c_x}, y_1, \ldots, y_{c_y}\}$ such that for all finite structures $\mathcal{A}$ of type $(r_1, \ldots, r_t)$:

$$\text{opt}_{\mathcal{Q}}(\mathcal{A}) = \max_{\mathcal{S}} |\{\mathbf{x} \in A^{c_x} : (\mathcal{A}, \mathcal{S}) \models (\exists \mathbf{y} \in A^{c_y}) : \psi(\mathbf{x}, \mathbf{y}, \mathcal{S})\}|.$$

Let s be the maximum number of occurrences of relations $S_1, \ldots, S_u$ in any disjunct of $\psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$. The standard parameterization p-$\mathcal{Q}$ of $\mathcal{Q}$ is the following problem:

Input: A finite structure $\mathcal{A}$ of type $(r_1, \ldots, r_t)$ and an integer k.
Parameter: k.
Task: Decide whether $\text{opt}_{\mathcal{Q}}(\mathcal{A}) \geq k$.

Similarly to the previous section, we consider the formula $\psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$ and a fixed instance $(\mathcal{A}, k)$ with $\mathcal{A} = (A, R_1, \ldots, R_t)$. We select tuples $\mathbf{x} \in A^{c_x}$ and $\mathbf{y} \in A^{c_y}$ and evaluate all literals of the form $R_i(\mathbf{z})$ and $\neg R_i(\mathbf{z})$ for some $\mathbf{z} \in \{x_1, \ldots, x_{x_c}, y_1, \ldots, y_{c_y}\}^{r_i}$. By checking whether $\mathbf{z} \in R_i$ we obtain 1 (true) or 0 (false) for each literal. Since $\psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$ is in disjunctive normal form, we delete all occurrences of 1 from the disjuncts and delete all disjuncts that contain a 0. Furthermore, we delete all disjuncts that contain contradicting literals $S_j(\mathbf{z}), \neg S_j(\mathbf{z})$ since they cannot be satisfied. We explicitly allow empty disjuncts that are satisfied by definition for the sake of simplicity (they occur when all literals in a disjunct are evaluated to 1). We obtain an equivalent formula that we denote with $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$.

Remark 4.1. For all $\mathbf{x}$, $\mathbf{y}$, and $\mathcal{S}$ it holds that $(\mathcal{A}, \mathcal{S}) \models \psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$ iff $(\mathcal{A}, \mathcal{S}) \models \psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$. Moreover, we can compute all formulas $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ for $\mathbf{x} \in A^{c_x}$, $\mathbf{y} \in A^{c_y}$ in polynomial time, since c_x, c_y, and the length of $\psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$ are constants independent of $\mathcal{A}$.

Definition 4.2. Let $\mathcal{A} = (A, R_1, \ldots, R_t)$ be a finite structure of type $(r_1, \ldots, r_t)$.
(a) We define $X_{\mathcal{A}} \subseteq A^{c_x}$ as the set of all tuples $\mathbf{x}$ such that $(\exists \mathbf{y}) : \psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ holds for some $\mathcal{S}$:

$$X_{\mathcal{A}} = \{\mathbf{x} : (\exists \mathcal{S}) : (\mathcal{A}, \mathcal{S}) \models (\exists \mathbf{y}) : \psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})\}.$$

(b) For $\mathbf{x} \in A^{c_x}$ we define $Y_{\mathcal{A}}(\mathbf{x})$ as the set of all tuples $\mathbf{y}$ such that $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ holds for some $\mathcal{S}$:

$$Y_{\mathcal{A}}(\mathbf{x}) = \{\mathbf{y} : (\exists \mathcal{S}) : (\mathcal{A}, \mathcal{S}) \models \psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})\}.$$

Remark 4.3. The sets $X_{\mathcal{A}}$ and $Y_{\mathcal{A}}(\mathbf{x})$ can be computed in polynomial time since the number of tuples $\mathbf{x} \in A^{c_x}$ and $\mathbf{y} \in A^{c_y}$ is polynomial in the size of A and $\psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$ is of constant length independent of $\mathcal{A}$.

Lemma 4.4. *Let $(\mathcal{A}, k)$ be an instance of p-$\mathcal{Q}$. If $|X_{\mathcal{A}}| \geq k \cdot 2^s$ then $\text{opt}_{\mathcal{Q}}(\mathcal{A}) \geq k$, i.e. $(\mathcal{A}, k)$ is a yes-instance.*

Proof. The lemma can be concluded from the proof of the constant-factor approximability of problems in MAX NP in [23]. For each $\mathbf{x} \in X_{\mathcal{A}}$ we fix a tuple $\mathbf{y} \in Y_{\mathcal{A}}(\mathbf{x})$ such that $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ is satisfiable. This yields $|X_{\mathcal{A}}|$ formulas, say $\psi_1, \ldots, \psi_{|X_{\mathcal{A}}|}$. Papadimitriou and Yannakakis

showed that one can efficiently compute an assignment that satisfies at least $\sum f_i$ of these formulas, where f_i is the fraction of all assignments that satisfies ψ_i.

To see that $f_i \geq 2^{-s}$; consider such a formula ψ_i. Since ψ_i is satisfiable there exists a satisfiable disjunct. To satisfy a disjunct of at most s literals, at most s variables need to be assigned accordingly. Since the assignment to all other variables can be arbitrary this implies that $f_i \geq 2^{-s}$. Thus we have that $\sum f_i \geq |X_{\mathcal{A}}| \cdot 2^{-s}$. Therefore $|X_{\mathcal{A}}| \geq k \cdot 2^s$ implies that the assignment satisfies at least k formulas, i.e. that $\text{opt}_{\mathcal{Q}}(\mathcal{A}) \geq k$. ∎

Henceforth we assume that $|X_{\mathcal{A}}| < k \cdot 2^s$.

Definition 4.5. Let $(\mathcal{A}, k)$ be an instance of p-$\mathcal{Q}$ with $\mathcal{A} = (A, R_1, \ldots, R_t)$. For $\mathbf{x} \in A^{c_x}$ we define $D_{\mathcal{A}}(\mathbf{x})$ as the set of all disjuncts of $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ for $\mathbf{y} \in Y_{\mathcal{A}}(\mathbf{x})$.

Definition 4.6. We define the *intersection of two disjuncts* as the conjunction of all literals that occur in both disjuncts. A *sunflower of a set of disjuncts* is a subset such that each pair of disjuncts in the subset has the same intersection (modulo permutation of the literals).

Remark 4.7. The size of each $D_{\mathcal{A}}(\mathbf{x})$ is bounded by the size of $Y_{\mathcal{A}}(\mathbf{x}) \subseteq A^{c_y}$ times the number of disjuncts of $\psi(\mathbf{x}, \mathbf{y}, \mathcal{S})$ which is a constant independent of $\mathcal{A}$. Thus the size of each $D_{\mathcal{A}}(\mathbf{x})$ is bounded by a polynomial in the input size. The definition of intersection and sunflowers among disjuncts is a direct analog that treats disjuncts as sets of literals.

Definition 4.8. A *partial assignment* is a set L of literals such that no literal is the negation of another literal in L. A formula is *satisfiable under* L if there exists an assignment that satisfies the formula and each literal in L.

Proposition 4.9. *Let $(\mathcal{A}, k)$ be an instance of p-$\mathcal{Q}$. For each $\mathbf{x} \in A^{c_x}$ there exists a set $D^*_{\mathcal{A}}(\mathbf{x}) \subseteq D_{\mathcal{A}}(\mathbf{x})$ of cardinality $O(k^s)$ such that:*

1. *For every partial assignment L of at most sk literals, $D^*_{\mathcal{A}}(\mathbf{x})$ contains a disjunct satisfiable under L, if and only if $D_{\mathcal{A}}(\mathbf{x})$ contains a disjunct satisfiable under L.*
2. *$D^*_{\mathcal{A}}(\mathbf{x})$ can be computed in time polynomial in $|\mathcal{A}|$.*

Proof. Let $\mathcal{A} = (A, R_1, \ldots, R_t)$ be a finite structure of type $(r_1, \ldots, r_t)$, let $\mathbf{x} \in A^{c_x}$, and let $D_{\mathcal{A}}(\mathbf{x})$ be a set of disjuncts according to Definition 4.5. From the Sunflower Corollary we can derive a polynomial-time algorithm that computes a set $D^*_{\mathcal{A}}(\mathbf{x})$ by successively shrinking sunflowers. We start by setting $D^*_{\mathcal{A}}(\mathbf{x}) = D_{\mathcal{A}}(\mathbf{x})$ and apply the following step while the cardinality of $D^*_{\mathcal{A}}(\mathbf{x})$ is greater than $(sk+1)^s \cdot s! \cdot s$.

We compute a sunflower of cardinality $sk+2$, say $F = \{f_1, \ldots, f_{sk+2}\}$, in time polynomial in $|D^*_{\mathcal{A}}(\mathbf{x})|$ (Sunflower Corollary). We delete a disjunct of F, say f_{sk+2}, from $D^*_{\mathcal{A}}(\mathbf{x})$. Let O and P be copies of $D^*_{\mathcal{A}}$ before respectively after deleting f_{sk+2}. Observe that $F' = F \setminus \{f_{sk+2}\}$ is a sunflower of cardinality $sk+1$ in P. Let L be a partial assignment of at most sk literals and assume that no disjunct in P is satisfiable under L. This means that for each disjunct of P there is a literal in L that contradicts it, i.e. a literal that is the negation of a literal in the disjunct. We focus on the sunflower F' in P. There must be a literal in L, say l, that contradicts at least two disjuncts of F', say f and f', since $|F'| = sk+1$ and $|L| \leq sk$. Therefore l is the negation of a literal in the intersection of f and f', i.e. the core of F'. Thus l contradicts also f_{sk+2} and we conclude that no disjunct in $O = P \cup \{f_{sk+2}\}$ is satisfiable under the partial assignment L. The reverse argument holds since all disjuncts of P are contained in O. Thus each step maintains the desired property (1).

At the end $D^*_{\mathcal{A}}(x)$ contains no more than $(sk+1)^s \cdot s! \cdot s \in O(k^s)$ disjuncts. For each $\mathbf{x}$ this takes time polynomial in the size of the input since the cardinality of $D_{\mathcal{A}}(\mathbf{x})$ is bounded by a polynomial in the input size and a disjunct is deleted in each step. ∎

Lemma 4.10. *Let $D'_{\mathcal{A}}(\mathbf{x})$ be a subset of $D_{\mathcal{A}}(\mathbf{x})$ such that $D^*_{\mathcal{A}}(\mathbf{x}) \subseteq D'_{\mathcal{A}}(\mathbf{x}) \subseteq D_{\mathcal{A}}(\mathbf{x})$. For any partial assignment L of at most sk literals it holds that $D_{\mathcal{A}}(\mathbf{x})$ contains a disjunct satisfiable under L if and only if $D'_{\mathcal{A}}(\mathbf{x})$ contains a disjunct satisfiable under L.*

Proof. Let L be a partial assignment of at most sk literals. If $D_{\mathcal{A}}(\mathbf{x})$ contains a disjunct satisfiable under L, then, by Proposition 4.9, this holds also for $D^*_{\mathcal{A}}(\mathbf{x})$. For $D^*_{\mathcal{A}}(\mathbf{x})$ and $D'_{\mathcal{A}}$ this holds since $D^*_{\mathcal{A}}(\mathbf{x}) \subseteq D'_{\mathcal{A}}(\mathbf{x})$. The same is true for $D'_{\mathcal{A}}(\mathbf{x})$ and $D_{\mathcal{A}}(\mathbf{x})$. ∎

Theorem 4.11. *Let $\mathcal{Q} \in$ MAX NP. The standard parameterization p-$\mathcal{Q}$ of $\mathcal{Q}$ admits a polynomial kernelization.*

Proof. The proof is organized in three parts. First, given an instance $(\mathcal{A}, k)$ of p-$\mathcal{Q}$, we construct an instance $(\mathcal{A}', k)$ of p-$\mathcal{Q}$ in time polynomial in the size of $(\mathcal{A}, k)$. In the second part, we prove that $(\mathcal{A}, k)$ and $(\mathcal{A}', k)$ are equivalent. In the third part, we conclude the proof by showing that the size of $(\mathcal{A}', k)$ is bounded by a polynomial in k.
(I.) Let $(\mathcal{A}, k)$ be an instance of p-$\mathcal{Q}$. We use the sets $D_{\mathcal{A}}(\mathbf{x})$ and $D^*_{\mathcal{A}}(\mathbf{x})$ according to Definition 4.5 and Proposition 4.9. Recall that $D_{\mathcal{A}}(\mathbf{x})$ is the set of all disjuncts of $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ for $\mathbf{y} \in Y_{\mathcal{A}}(\mathbf{x})$. Thus, for each disjunct $d \in D^*_{\mathcal{A}}(\mathbf{x}) \subseteq D_{\mathcal{A}}(\mathbf{x})$, we can select a $\mathbf{y}_d \in Y_{\mathcal{A}}(\mathbf{x})$ such that d is a disjunct of $\psi_{\mathbf{x},\mathbf{y}_d}(\mathcal{S})$. Let $Y'_{\mathcal{A}}(\mathbf{x}) \subseteq Y_{\mathcal{A}}(\mathbf{x})$ be the set of these selected tuples $\mathbf{y}_d$. Let $D'_{\mathcal{A}}(\mathbf{x})$ be the set of all disjuncts of $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ for $\mathbf{y} \in Y'_{\mathcal{A}}(\mathbf{x})$. Since $D^*_{\mathcal{A}}(\mathbf{x})$ contains some disjuncts of $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ for $\mathbf{y} \in Y'_{\mathcal{A}}(\mathbf{x})$ and $D_{\mathcal{A}}(\mathbf{x})$ contains all disjuncts of $\psi_{\mathbf{x},\mathbf{y}}(\mathcal{S})$ for $\mathbf{y} \in Y_{\mathcal{A}}(\mathbf{x}) \supseteq Y'_{\mathcal{A}}(\mathbf{x})$, we have that $D^*_{\mathcal{A}}(\mathbf{x}) \subseteq D'_{\mathcal{A}}(\mathbf{x}) \subseteq D_{\mathcal{A}}(\mathbf{x})$.

For each $\mathbf{x}$ this takes time $O(|D^*_{\mathcal{A}}(\mathbf{x})| \cdot |Y^*_{\mathcal{A}}(\mathbf{x})|) \subseteq O(k^s \cdot |A|^{c_y})$. Computing $Y'_{\mathcal{A}}(\mathbf{x})$ for all $\mathbf{x} \in A^{c_x}$ takes time $O(|A|^{c_x} \cdot k^s \cdot |A|^{c_y})$, i.e. time polynomial in the size of $(\mathcal{A}, k)$ since k is never larger than $|A|^{c_x}$.[4]

Let $A' \subseteq A$ be the set of all components of $\mathbf{x} \in X_{\mathcal{A}}$ and $\mathbf{y} \in Y'_{\mathcal{A}}(\mathbf{x})$ for all $\mathbf{x} \in X_{\mathcal{A}}$. This ensures that $X_{\mathcal{A}} \subseteq (A')^{c_x}$ and $Y'_{\mathcal{A}}(\mathbf{x}) \subseteq (A')^{c_y}$ for all $\mathbf{x} \in X_{\mathcal{A}}$. Let R'_i be the restriction of R_i to A' and let $\mathcal{A}' = (A', R'_1, \ldots, R'_t)$.
(II.) We will now prove that $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) \geq k$ if and only if $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}') \geq k$, i.e. that $(\mathcal{A}, k)$ and $(\mathcal{A}', k)$ are equivalent. Assume that $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) \geq k$ and let $\mathcal{S} = (S_1, \ldots, S_u)$ such that $|\{\mathbf{x} : (\mathcal{A}, \mathcal{S}) \models (\exists \mathbf{y}) : \psi(\mathbf{x}, \mathbf{y}, \mathcal{S})\}| \geq k$. This implies that there must exist tuples $\mathbf{x}_1, \ldots, \mathbf{x}_k \in A^{c_x}$ and $\mathbf{y}_1, \ldots, \mathbf{y}_k \in A^{c_y}$ such that $\mathcal{S}$ satisfies $\psi_{\mathbf{x}_i,\mathbf{y}_i}(\mathcal{S})$ for $i = 1, \ldots, k$. Thus $\mathcal{S}$ must satisfy at least one disjunct in each $\psi_{\mathbf{x}_i,\mathbf{y}_i}(\mathcal{S})$ since these formulas are in disjunctive normal form. Accordingly let $d_1, \ldots, d_k$ be disjuncts such that $\mathcal{S}$ satisfies the disjunct d_i in $\psi_{\mathbf{x}_i,\mathbf{y}_i}(\mathcal{S})$ for $i = 1, \ldots, k$. We show that there exists $\mathcal{S}'$ such that:

$$|\{\mathbf{x} : (\mathcal{A}', \mathcal{S}') \models (\exists \mathbf{y}) : \psi(\mathbf{x}, \mathbf{y}, \mathcal{S}')\}| \geq k.$$

For $p = 1, \ldots, k$ we apply the following step: If $\mathbf{y}_p \in Y'_{\mathcal{A}}(\mathbf{x}_p)$ then do nothing. Otherwise consider the partial assignment L consisting of the at most sk literals of the disjuncts $d_1, \ldots, d_k$. The set $D_{\mathcal{A}}(\mathbf{x}_p)$ contains a disjunct that is satisfiable under L, namely d_p. By Lemma 4.10, it follows that $D'_{\mathcal{A}}(\mathbf{x}_p)$ also contains a disjunct satisfiable under L, say d'_p. Let $\mathbf{y}'_p \in Y'_{\mathcal{A}}(\mathbf{x}_p)$ such that d'_p is a disjunct of $\psi_{\mathbf{x}_p,\mathbf{y}'_p}(\mathcal{S})$. Such a $\mathbf{y}'_p$ can be found by selection of $D'_{\mathcal{A}}(\mathbf{x}_p)$. Change $\mathcal{S}$ in the following way to satisfy the disjunct d'_p. For each literal of d'_p of the form $S_i(\mathbf{z})$ add $\mathbf{z}$ to the relation S_i. Similarly for each literal of the form $\neg S_i(\mathbf{z})$

[4] That is, $(\mathcal{A}, k)$ is a no-instance if $k > |A|^{c_x}$ since k exceeds the number of tuples $\mathbf{x} \in A^{c_x}$.

remove $\mathbf{z}$ from S_i. This does not change the fact that $\mathcal{S}$ satisfies the disjunct d_i in $\psi_{\mathbf{x}_i,\mathbf{y}_i}(\mathcal{S})$ for $i = 1, \ldots, k$ since, by selection, d'_p is satisfiable under L. Then we replace $\mathbf{y}_p$ by $\mathbf{y}'_p$ and d_p by d'_p. Thus we maintain that $\mathcal{S}$ satisfies d_i in $\psi_{\mathbf{x}_i,\mathbf{y}_i}(\mathcal{S})$ for $i = 1, \ldots, k$.

After these steps we obtain $\mathcal{S}$ as well as tuples $\mathbf{x}_1, \ldots, \mathbf{x}_k$, $\mathbf{y}_1, \ldots, \mathbf{y}_k$ with $\mathbf{y}_i \in Y'_{\mathcal{A}}(\mathbf{x}_i)$, and disjuncts $d_1, \ldots, d_k$ such that $\mathcal{S}$ satisfies d_i in $\psi_{\mathbf{x}_i,\mathbf{y}_i}(\mathcal{S})$ for $i = 1, \ldots, k$. Let $\mathcal{S}'$ be the restriction of $\mathcal{S}$ to A'. Then we have that $(\mathcal{A}', \mathcal{S}') \models \psi_{\mathbf{x}_i,\mathbf{y}_i}(\mathcal{S}')$ for $i = 1, \ldots, k$ since A' is defined to contain the components of tuples $\mathbf{x} \in X_{\mathcal{A}}$ and of all tuples $\mathbf{y} \in Y'_{\mathcal{A}}(\mathbf{x})$ for $\mathbf{x} \in X_{\mathcal{A}}$. Hence $\mathbf{x}_i \in \{\mathbf{x} : (\mathcal{A}', \mathcal{S}') \models (\exists \mathbf{y}) : \psi(\mathbf{x}, \mathbf{y}, \mathcal{S}')\}$ for $i = 1, \ldots, k$. Thus $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}') \geq k$.

For the reverse direction assume that $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}') \geq k$. Since $A' \subseteq A$ it follows that

$$\{\mathbf{x} : (\mathcal{A}', \mathcal{S}') \models (\exists \mathbf{y}) : \psi(\mathbf{x}, \mathbf{y}, \mathcal{S}')\} \subseteq \{\mathbf{x} : (\mathcal{A}, \mathcal{S}') \models (\exists \mathbf{y}) : \psi(\mathbf{x}, \mathbf{y}, \mathcal{S}')\}.$$

Thus $|\{\mathbf{x} : (\mathcal{A}, \mathcal{S}') \models (\exists \mathbf{y}) : \psi(\mathbf{x}, \mathbf{y}, \mathcal{S}')\}| \geq k$, implying that $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) \geq k$. Therefore $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) \geq k$ if and only if $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}') \geq k$. Hence $(\mathcal{A}, k)$ and $(\mathcal{A}', k)$ are equivalent instances of p-$\mathcal{Q}$.

(III.) We conclude the proof by providing an upper bound on the size of $(\mathcal{A}', k)$ that is polynomial in k. For the sets $Y'_{\mathcal{A}}(\mathbf{x})$ we selected one tuple $\mathbf{y}$ for each disjunct in $D^*_{\mathcal{A}}(\mathbf{x})$. Thus $|Y'_{\mathcal{A}}(\mathbf{x})| \leq |D^*(\mathbf{x})| \in O(k^s)$ for all $\mathbf{x} \in X_{\mathcal{A}}$. The set A' contains the components of tuples $\mathbf{x} \in X_{\mathcal{A}}$ and of all tuples $\mathbf{y} \in Y'_{\mathcal{A}}(\mathbf{x})$ for $\mathbf{x} \in X_{\mathcal{A}}$. Thus

$$\begin{aligned} |A'| &\leq c_x \cdot |X_{\mathcal{A}}| + c_y \cdot \textstyle\sum_{\mathbf{x} \in X_{\mathcal{A}}} |Y'_{\mathcal{A}}(\mathbf{x})| \\ &\leq c_x \cdot |X_{\mathcal{A}}| + c_y \cdot |X_{\mathcal{A}}| \cdot O(k^s) \\ &< c_x \cdot k \cdot 2^s + c_y \cdot k \cdot 2^s \cdot O(k^s) = O(k^{s+1}). \end{aligned}$$

For each relation R'_i we have $|R'_i| \leq |A'|^{r_i} \in O(k^{(s+1)r_i})$. Thus the size of $(\mathcal{A}', k)$ is bounded by $O(k^{(s+1)m})$, where m is the largest arity of a relation R_i. ∎

Remark 4.12. For MAX SNP one can prove a stronger result that essentially relies on Lemma 4.4. That way one obtains bounds for the sizes of A' and $(\mathcal{A}', k)$ of $O(k)$ and $O(k^m)$ respectively.

5. Conclusion

We have constructively established that the standard parameterizations of problems in MIN $\mathrm{F}^+\Pi_1$ and MAX NP admit polynomial kernelizations. Thus a strong relation between constant-factor approximability and polynomial kernelizability has been showed for two large classes of problems. It remains an open problem to give a more general result that covers all known examples (e.g. Feedback Vertex Set). It might be profitable to consider closures of MAX SNP under reductions that preserve constant-factor approximability. Khanna et al. [16] proved that APX and APX-PB are the closures of MAX SNP under PTAS-preserving reductions and E-reductions, respectively. Since both classes contain Bin Packing which does not admit a polynomial kernelization, this leads to the question whether polynomial kernelizability or fixed-parameter tractability are maintained under restricted versions of these reductions.

Acknowledgement

We would like to thank Iyad Kanj for pointing out Bin Packing as an example that is constant-factor approximable but does not admit a polynomial kernelization.

References

[1] Faisal N. Abu-Khzam. Kernelization algorithms for d-hitting set problems. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *WADS*, volume 4619 of *LNCS*, pages 434–445. Springer, 2007.

[2] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[3] Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 12(3):289–297, 1999.

[4] Hans L. Bodlaender. A cubic kernel for feedback vertex set. In Wolfgang Thomas and Pascal Weil, editors, *STACS*, volume 4393 of *LNCS*, pages 320–331. Springer, 2007.

[5] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels (extended abstract). In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (1)*, volume 5125 of *LNCS*, pages 563–574. Springer, 2008.

[6] Liming Cai and Jianer Chen. On fixed-parameter tractability and approximability of NP optimization problems. *J. Comput. Syst. Sci.*, 54(3):465–474, 1997.

[7] Liming Cai and Xiuzhen Huang. Fixed-parameter approximation: Conceptual framework and approximability results. In Hans L. Bodlaender and Michael A. Langston, editors, *IWPEC*, volume 4169 of *Lecture Notes in Computer Science*, pages 96–108. Springer, 2006.

[8] Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

[9] Rod G. Downey and M. R. Fellows. *Parameterized Complexity (Monographs in Computer Science)*. Springer, November 1998.

[10] Paul Erdős and Richard Rado. Intersection theorems for systems of sets. *J. London Math. Soc.*, 35:85–90, 1960.

[11] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 43–73, SIAM, Philadelphia, PA, 1974.

[12] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008.

[13] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, March 2006.

[14] Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In Richard E. Ladner and Cynthia Dwork, editors, *STOC*, pages 133–142. ACM, 2008.

[15] Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.*, 31(5):1608–1623, 2002.

[16] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh V. Vazirani. On syntactic versus computational views of approximability. *SIAM J. Comput.*, 28(1):164–191, 1998.

[17] Phokion G. Kolaitis and Madhukar N. Thakur. Logical definability of NP optimization problems. *Inf. Comput.*, 115(2):321–353, 1994.

[18] Phokion G. Kolaitis and Madhukar N. Thakur. Approximation properties of NP minimization classes. *J. Comput. Syst. Sci.*, 50(3):391–411, 1995.

[19] Assaf Natanzon, Ron Shamir, and Roded Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM J. Comput.*, 30(4):1067–1079, 2000.

[20] Rolf Niedermeier. *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford University Press, USA, March 2006.

[21] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. Smaller kernels for hitting set problems of constant arity. In Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne, editors, *IWPEC*, volume 3162 of *LNCS*, pages 121–126. Springer, 2004.

[22] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, November 1993.

[23] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 613–624
www.stacs-conf.org

LOCAL MULTICOLORING ALGORITHMS:
COMPUTING A NEARLY-OPTIMAL TDMA SCHEDULE IN CONSTANT TIME

FABIAN KUHN [1]

[1] MIT, Computer Science and Artificial Intelligence Lab
32 Vassar St, Cambridge, MA 02139, USA
E-mail address: fkuhn@csail.mit.edu

ABSTRACT. We are given a set V of autonomous agents (e.g. the computers of a distributed system) that are connected to each other by a graph $G = (V, E)$ (e.g. by a communication network connecting the agents). Assume that all agents have a unique ID between 1 and N for a parameter $N \geq |V|$ and that each agent knows its ID as well as the IDs of its neighbors in G. Based on this limited information, every agent v must autonomously compute a set of colors $S_v \subseteq C$ such that the color sets S_u and S_v of adjacent agents u and v are disjoint. We prove that there is a deterministic algorithm that uses a total of $|C| = \mathcal{O}(\Delta^2 \log(N)/\varepsilon^2)$ colors such that for every node v of G (i.e., for every agent), we have $|S_v| \geq |C|\cdot(1-\varepsilon)/(\delta_v+1)$, where δ_v is the degree of v and where Δ is the maximum degree of G. For $N = \Omega(\Delta^2 \log \Delta)$, $\Omega(\Delta^2 + \log\log N)$ colors are necessary even to assign at least one color to every node (i.e., to compute a standard vertex coloring). Using randomization, it is possible to assign an $(1-\varepsilon)/(\delta+1)$-fraction of all colors to every node of degree δ using only $\mathcal{O}(\Delta \log|V|/\varepsilon^2)$ colors w.h.p. We show that this is asymptotically almost optimal. For graphs with maximum degree $\Delta = \Omega(\log|V|)$, $\Omega(\Delta \log|V|/\log\log|V|)$ colors are needed in expectation, even to compute a valid coloring.

The described multicoloring problem has direct applications in the context of wireless ad hoc and sensor networks. In order to coordinate the access to the shared wireless medium, the nodes of such a network need to employ some medium access control (MAC) protocol. Typical MAC protocols control the access to the shared channel by time (TDMA), frequency (FDMA), or code division multiple access (CDMA) schemes. Many channel access schemes assign a fixed set of time slots, frequencies, or (orthogonal) codes to the nodes of a network such that nodes that interfere with each other receive disjoint sets of time slots, frequencies, or code sets. Finding a valid assignment of time slots, frequencies, or codes hence directly corresponds to computing a multicoloring of a graph G. The scarcity of bandwidth, energy, and computing resources in ad hoc and sensor networks, as well as the often highly dynamic nature of these networks require that the multicoloring can be computed based on as little and as local information as possible.

1. Introduction

In this paper, we look at a variant of the standard vertex coloring problem that we name graph *multicoloring*. Given an n-node graph $G = (V, E)$, the goal is to assign a set S_v of colors to each node $v \in V$ such that the color sets S_u and S_v of two adjacent nodes $u \in V$ and $v \in V$ are disjoint while at the same time, the fraction of colors assigned to each node is as large as possible and the

Key words and phrases: distributed algorithms, graph coloring, local algorithms, medium access control, multicoloring, TDMA, wireless networks.

For space reasons, most proofs are omitted from this extended abstract. A full version can be received from the author's web site at http://people.csail.mit.edu/fkuhn/publications/multicoloring.pdf.

total number of colors used is as small as possible. In particular, we look at the following *distributed* variant of this multicoloring problem. Each node has a unique identifier (ID) between 1 and N for an integer parameter $N \geq n$. The nodes are *autonomous agents* and we assume that every agent has only very limited, *local* information about G. Specifically, we assume that every node $v \in V$ merely knows its own ID as well as the IDs of all its neighbors. Based on this local information, every node v needs to compute a color set S_v such that the color sets computed by adjacent nodes are disjoint. Since our locality condition implies that every node is allowed to communicate with each neighbor only once, we call such a a distributed algorithm a *one-shot algorithm*.

We prove nearly tight upper and lower bounds for deterministic and randomized algorithms solving the above distributed multicoloring problem. Let Δ be the largest degree of G. We show that for every $\varepsilon \in (0,1)$, there is a deterministic multicoloring algorithm that uses $\mathcal{O}(\Delta^2 \log(N)/\varepsilon^2)$ colors and assigns a $(1-\varepsilon)/(\delta+1)$-fraction of all colors to each node of degree δ. Note that because a node v of degree δ does not know anything about the topology of G (except that itself has δ neighbors), no one-shot multicoloring algorithm can assign more than a $1/(\delta+1)$-fraction of the colors to all nodes of degree δ (the nodes could be in a clique of size $\delta+1$). The upper bound proof is based on the probabilistic method and thus only establishes the existence of an algorithm. We describe an algebraic construction yielding an explicit algorithm that achieves the same bounds up to polylogarithmic factors. Using $\mathcal{O}(\Delta^2 \log^2 N)$ colors, for a value $\varepsilon > 0$, the algorithm assigns a $\varepsilon/\mathcal{O}(\delta^{1+\varepsilon} \log N)$-fraction of all colors to nodes of degree δ. At the cost of using $\mathcal{O}(\Delta^{\log^* N} \log N)$ colors, it is even possible to improve the fraction of colors assigned to each node by a factor of $\log N$. The deterministic upper bound results are complemented by a lower bound showing that if $N = \Omega(\Delta^2 \log \Delta)$, even for the standard vertex coloring problem, every deterministic one-shot algorithm needs to use at least $\Omega(\Delta^2 + \log\log N)$ colors.

If we allow the nodes to use randomization (and only require that the claimed bounds are obtained with high probability), we can do significantly better. In a randomized one-shot algorithm, we assume that every node can compute a sequence of random bits at the beginning of an algorithm and that nodes also know their own random bits as well as the random bits of the neighbors when computing the color set. We show that for $\varepsilon \in (0,1)$, with high probability, $\mathcal{O}(\Delta \log(n)/\varepsilon^2)$ colors suffice to assign a $(1-\varepsilon)/(\delta+1)$-fraction of all colors to every node of degree δ. If $\log n \leq \Delta \leq n^{1-\varepsilon}$ for a constant $\varepsilon > 0$, we show that every randomized one-shot algorithm needs at least $\Omega(\Delta \log n/\log\log n)$ colors. Again, the lower bound even holds for standard vertex coloring algorithms where every node only needs to choose a single color.

Synchronizing the access to a common resource is a typical application of coloring in networks. If we have a c-coloring of the network graph, we can partition the resource (and/or time) into c parts and assign a part to each node v depending on v's color. In such a setting, it seems natural to use a multicoloring instead of a standard vertex coloring and assign more than one part of the resource to every node. This allows to use the resource more often and thus more efficiently.

The most prominent specific example of this basic approach occurs in the context of media access control (MAC) protocols for wireless ad hoc and sensor networks. These networks consist of autonomous wireless devices that communicate with each other by the use of radio signals. If two or more close-by nodes transmit radio signals at the same time, a receiving node only hears the superposition of all transmitted signals. Hence, simultaneous transmissions of close-by nodes interfere with each other and we thus have to control the access to the wireless channel. A standard way to avoid interference between close-by transmissions is to use a time (TDMA), frequency (FDMA), or code division multiple access (CDMA) scheme to divide the channel among the nodes. A TDMA protocol divides the time into time slots and assigns different time slots to conflicting nodes. When using FDMA, nodes that can interfere with each other are assigned different frequencies, whereas a CDMA scheme uses different (orthogonal) codes for interfering nodes. Classically,

TDMA, FDMA, and CDMA protocols are implemented by a standard vertex coloring of the graph induced by the interference relations. In all three cases, it would be natural to use the more general multicoloring problem in order to achieve a more effective use of the wireless medium. Efficient TDMA schedules, FDMA frequency assignments, or CDMA code assignments are all directly obtained from a multicoloring of the interference graph where the fraction of colors assigned to each nodes is as large as possible. It is also natural to require that the total number of colors is small. This keeps the length of a TDMA schedule or the total number of frequencies or codes small and thus helps to improve the efficiency and reduce unnecessary overhead of the resulting MAC protocols.

In contrast to many wired networks, wireless ad hoc and sensor networks typically consist of small devices that have limited computing and storage capabilities. Because these devices operate on batteries, wireless nodes also have to keep the amount of computation and especially communication to a minimum in order to save energy and thus increase their lifetime. As the nodes of an ad hoc or sensor network need to operate without central control, everything that is computed, has to be computed by a distributed algorithm by the nodes themselves. Coordination between the nodes is achieved by exchanging messages. Because of the resource constraints, these distributed algorithms need to be as simple and efficient as possible. The messages transmitted and received by each node should be as few and as short as possible. Note that because of interference, the bandwidth of each local region is extremely limited. Typically, for a node v, the time needed to even receive a single message from all neighbors is proportional to the degree of v (see e.g. [19]). As long as the information provided to each node is symmetric, it is clear that every node needs to know the IDs of all adjacent nodes in G in order to compute a reasonably good multicoloring of G. Hence, the one-shot multicoloring algorithms considered in this paper base their computations on the minimum information needed to compute a non-trivial solution to the problem. Based on the above observations, even learning the IDs of all neighbors requires quite a bit of time and resources. Hence, acquiring significantly more information might already render an algorithm inapplicable in practice.[1]

As a result of the scarcity of resources, the size and simplicity of the wireless devices used in sensor networks, and the dependency of the characteristic of radio transmissions on environmental conditions, ad hoc and sensor networks are much less stable than usual wired networks. As a consequence, the topology of these networks (and of their interference graph) can be highly dynamic. This is especially true for ad hoc networks, where it is often even assumed that the nodes are mobile and thus can move in space. In order to adapt to such dynamic conditions, a multicoloring needs to be recomputed periodically. This makes the resource and time efficiency of the used algorithms even more important. This is particularly true for the locality of the algorithms. If the computation of every node only depends on the topology of a close-by neighborhood, dynamic changes also only affect near-by nodes.

The remainder of the paper is organized as follows. In Section 2, we discuss related work. The problem is formally defined in Section 3. We present the deterministic and randomized upper bounds in Section 4 and the lower bounds in Section 5.

2. Related Work

There is a rich literature on distributed algorithms to compute classical vertex colorings (see e.g. [1, 4, 11, 15, 16, 21]). The paper most related to the present one is [15]. In [15], deterministic algorithms for the standard coloring problem in the same distributed setting are studied (i.e., every

[1] It seems that in order to achieve a significant improvement on the multicolorings computed by the algorithms presented in this paper, every node would need much more information. Even if every node knows its complete $O(\log \Delta)$-neighborhood, the best deterministic coloring algorithm that we are aware of needs $\Theta(\Delta^2)$ colors.

node has to compute its color based on its ID and the IDs of its neighbors). The main result is a $\Omega(\Delta^2/\log^2\Delta)$ lower bound on the number of colors. The first paper to study distributed coloring is a seminal paper by Linial [16]. The main result of [16] is an $\Omega(\log^* n)$-time lower bound for coloring a ring with a constant number of colors. As a corollary of this lower bound, one obtains an $\Omega(\log\log N)$ lower bound on the number of colors for deterministic one-shot coloring algorithms as studied in this paper. Linial also looks at distributed coloring algorithms for general graph and shows that one can compute an $\mathcal{O}(\Delta^2)$-coloring in time $\mathcal{O}(\log^* n)$. In order to color a general graph with less colors, the best known distributed algorithms are significantly slower.[2] Using randomization, an $\mathcal{O}(\Delta)$-coloring can be obtained in time $\mathcal{O}(\sqrt{\log n})$ [14]. Further, the fastest algorithm to obtain a $(\Delta+1)$-coloring is based on an algorithm to compute a maximal independent set by Luby [17] and on a reduction described in [16] and has time complexity $\mathcal{O}(\log n)$. The best known deterministic algorithms to compute a $(\Delta+1)$-coloring have time complexities $2^{\mathcal{O}(\sqrt{\log n})}$ and $\mathcal{O}(\Delta\log\Delta+\log^* n)$ and are described in [21] and [15], respectively. For special graph classes, there are more efficient deterministic algorithms. It has long been known that in rings [4] and bounded degree graphs [11, 16], a $(\Delta+1)$-coloring can be computed in time $\mathcal{O}(\log^* n)$. Very recently, it has been shown that this also holds for the much larger class of graphs with bounded local independent sets [26]. In particular, this graph class contains all graph classes that are typically used to model wireless ad hoc and sensor networks. Another recent result shows that graphs of bounded arboricity can be colored with a constant number of colors in time $\mathcal{O}(\log n)$ [3].

Closely related to vertex coloring algorithms are distributed algorithms to compute edge colorings [5, 12, 22]. In a seminal paper, Naor and Stockmeyer were the first to look at distributed algorithms where all nodes have to base their decisions on constant neighborhoods [20]. It is shown that a weak coloring with $f(\Delta)$ colors (every node needs to have a neighbor with a different color) can be computed in time 2 if every vertex has an odd degree. Another interesting approach is taken in [9] where the complexity of distributed coloring is studied in case there is an oracle that gives some nodes a few bits of extra information.

There are many papers that propose to use some graph coloring variant in order to compute TDMA schedules and FDMA frequency or CDMA code assignments (see e.g. [2, 10, 13, 18, 24, 25, 27]). Many of these papers compute a vertex coloring of the network graph such that nodes at distance at most 2 have different colors. This guarantees that no two neighbors of a node use the same time slot, frequency, or code. Some of the papers also propose to construct a TDMA schedule by computing an edge coloring and using different time slots for different edges. Clearly, it is straight-forward to use our algorithms for edge colorings, i.e., to compute a multicoloring of the line graph. With the exception of [13] all these papers compute a coloring and assign only one time slot, frequency, or code to every node or edge. In [13], first, a standard coloring is computed. Based on this coloring, an improved slot assignment is constructed such that in the end, the number of slots assigned to a node is inversely proportional to the number of colors in its neighborhood.

3. Formal Problem Description

3.1. Mathematical Preliminaries

Throughout the paper, we use $\log(\cdot)$ to denote logarithms to base 2 and $\ln(\cdot)$ to denote natural logarithms, respectively. By $\log^{(i)} x$ and by $\ln^{(i)} x$, we denote the i-fold applications of the logarithm functions $\log$ and $\ln$ to x, respectively[3]. The log star function is defined as $\log^* n :=$

[2] In [6], it is claimed that an $\mathcal{O}(\Delta)$ coloring can be computed in time $\mathcal{O}(\log^*(n/\Delta))$. However, the argumentation in [6] has a fundamental flaw that cannot be fixed [23].

[3] We have $\log^{(0)} x = \ln^{(0)} x = x$, $\log^{(i+1)} x = \log(\log^{(i)} x)$, and $\ln^{(i+1)} x = \ln(\ln^{(i)} x)$. Note that we also use $\log^i x = (\log x)^i$ and $\ln^i x = (\ln x)^i$

$\min_i\{\log^{(i)} n \leq 1\}$. We also use the following standard notations. For an integer $n \geq 1$, $[n] = \{1,\ldots,n\}$. For a finite set Ω and an integer $k \in \{0,\ldots,|\Omega|\}$, $\binom{\Omega}{k} = \{S \in 2^\Omega : |S| = k\}$. The term with high probability (w.h.p.) means with probability at least $1 - 1/n^c$ for a constant $c \geq 1$.

3.2. Multicoloring

The multicoloring problem that was introduced in Section 1 can be formally defined as follows.

Definition 3.1 (Multicoloring)**.** An $(\rho(\delta), k)$-multicoloring γ of a graph $G = (V, E)$ is a mapping $\gamma : V \to 2^{[k]}$ that assigns a set $\gamma(v) \subset [k]$ of colors to each node v of G such that $\forall\{u, v\} \in E : \gamma(u) \cap \gamma(v) = \emptyset$ and such that for every node $v \in V$ of degree δ, $|\gamma(v)|/k \geq \rho(\delta)/(\delta + 1)$.

We call $\rho(\delta)$ the *approximation ratio* of a $(\rho(\delta), k)$-multicoloring. Because in a one-shot algorithm (cf. the next section for a formal definition), a node of degree δ cannot distinguish G from $K_{\delta+1}$, the approximation ratio of every one-shot algorithm needs to be at most 1.

The multicoloring problem is related to the fractional coloring problem in the following way. Assume that every node is assigned the same number c of colors and that the total number of colors is k. Taking every color with fraction $1/c$ then leads to a fractional (k/c)-coloring of G. Hence, in this case, k/c is lower bounded by the fractional chromatic number $\chi_f(G)$ of G.

3.3. One-Shot Algorithms

As outlined in the introduction, we are interested in local algorithms to compute multicolorings of an n-node graph $G = (V, E)$. For a parameter $N \geq n$, we assume that every node v has a unique ID $x_v \in [N]$. In deterministic algorithms, every node has to compute a color set based on its own ID as well as the IDs of its neighbors. For randomized algorithms, we assume that nodes also know the random bits of their neighbors. Formally, a one-shot algorithm can be defined as follows.

Definition 3.2 (One-Shot Algorithm)**.** We call a distributed algorithm a one-shot algorithm if every node v performs (a subset of) the following three steps:

1. Generate sequence R_v of random bits (deterministic algorithms: $R_v = \emptyset$)
2. Send x_v, R_v to all neighbors
3. Compute solution based on x_v, R_v, and the received information

Assume that G is a network graph such that two nodes u and v can directly communicate with each other iff they are connected by an edge in G. In the standard *synchronous message passing* model, time is divided into rounds and in every round, every node of G can send a message to each of its neighbors. One-shot algorithms then exactly correspond to computations that can be carried out in a single communication round.

For deterministic one-shot algorithms, the output of every node v is a function of v's ID x_v and the IDs of v's neighbors. We call this information on which v bases its decisions, the *one-hop view* of v.

Definition 3.3 (One-Hop View)**.** Consider a node v with ID x_v and let Γ_v be the set of IDs of the neighbors of v. We call the pair (x_v, Γ_v) the one-hop view of v.

Let (x_u, Γ_u) and (x_v, Γ_v) be the one-hop views of two adjacent nodes. Because u and v are neighbors, we have $x_u \in \Gamma_v$ and that $x_v \in \Gamma_u$. It is also not hard to see that

$$\forall x_u, x_v \in [N] \text{ and } \forall \Gamma_u, \Gamma_v \in 2^{[N]} \text{ such that } x_u \neq x_v, x_u \in \Gamma_v \setminus \Gamma_u, x_v \in \Gamma_u \setminus \Gamma_v, \tag{3.1}$$

there is a labeled graph that has two adjacent nodes u and v with one-hop views (x_u, Γ_u) and (x_v, Γ_v), respectively. Assume that we are given a graph with maximum degree Δ (i.e., for all

one-hop views (x_v, Γ_v), we have $|\Gamma_v| \leq \Delta$). A one-shot vertex coloring algorithm maps every possible one-hop view to a color. A correct coloring algorithm must assign different colors to two one-hop views (x_u, Γ_u) and (x_v, Γ_v) iff they satisfy Condition (3.1). This leads to the definition of the *neighborhood graph* $\mathcal{N}_1(N, \Delta)$ [15] (the general notion of neighborhood graphs has been introduced in [16]). The nodes of $\mathcal{N}_1(N, \Delta)$ are all one-hop views (x_v, Γ_v) with $|\Gamma_v| \leq \Delta$. There is an edge between (x_u, Γ_u) and (x_v, Γ_v) iff the one-hop views satisfy Condition (3.1). Hence, a one-shot coloring algorithm must assign different colors to two one-hop views iff they are neighbors in $\mathcal{N}_1(N, \Delta)$. The number of colors that are needed to properly color graphs with maximum degree Δ by a one-shot algorithm therefore exactly equals the chromatic number $\chi\big(\mathcal{N}_1(N, \Delta)\big)$ of the neighborhood graph (see [15, 16] for more details). Similarly, a one-shot $(\rho(\delta), k)$-multicoloring algorithm corresponds to a $(\rho(\delta), k)$-multicoloring of the neighborhood graph.

4. Upper Bounds

In this section, we prove all the upper bounds claimed in Section 1. We first prove that an efficient deterministic one-shot multicoloring algorithm exists in Section 4.1. Based on similar ideas, we derive an almost optimal randomized algorithm in Section 4.2. Finally, in Section 4.3, we introduce constructive methods to obtain one-shot multicoloring algorithms. For all algorithms, we assume that the nodes know the size of the ID space N as well as Δ, an upper bound on the largest degree in the network. It certainly makes sense that nodes are aware of the used ID space. Note that it is straight-forward to see that there cannot be a non-trivial solution to the one-shot multicoloring problem if the nodes do not have an upper bound on the maximum degree in the network.

4.1. Existence of an Efficient Deterministic Algorithm

The existence of an efficient, deterministic one-shot multicoloring algorithm is established by the following theorem.

Theorem 4.1. *Assume that we are given a graph with maximum degree Δ and node IDs in $[N]$. Then, for all $0 < \varepsilon \leq 1$, there is a deterministic, one-shot $\big(1-\varepsilon, \mathcal{O}(\Delta^2 \log(N)/\varepsilon^2)\big)$-multicoloring algorithm.*

Proof. We use permutations to construct colors as described in [15]. For $i = 1, \ldots, k$, let $\prec_i$ be a global order on the ID set $[N]$. A node v with 1-hop view (x_v, Γ_v) includes color i in its color set iff $\forall y \in \Gamma_v : x_v \prec_i y$. It is clear that with this approach the color sets of adjacent nodes are disjoint. In order to show that nodes of degree δ obtain a $\rho/(\delta+1)$-fraction of all colors, we need to show that for all $\delta \in [\Delta]$, all $x \in [N]$, and all $\Gamma \in \binom{[N]\setminus\{x\}}{\delta}$, for all $y \in \Gamma$, $x \prec_i y$ for at least $k\rho/(\delta+1)$ global orders $\prec_i$. We use the probabilistic method to show that a set of size $k = 2(\Delta+1)^2 \ln(N)/\varepsilon^2$ of global orders $\prec_i$ exists such that every node of degree $\delta \in [\Delta]$ gets at least an $(1-\varepsilon)/(\delta+1)$-fraction of the k colors. Such a set implies that there exists an algorithm that satisfies the claimed bounds for all graphs with maximum degree Δ and IDs in $[N]$.

Let $\prec_1, \ldots, \prec_k$ be k global orders chosen independently and uniformly at random. The probability that a node v with degree δ and 1-hop view (x_v, Γ_v) gets color i is $1/(\delta+1)$ (note that $|\Gamma_v| = \delta$). Let X_v be the number of colors that v gets. We have $\mathbb{E}[X_v] = k/(\delta+1) \geq k/(\Delta+1)$. Using a Chernoff bound, we then obtain

$$\mathbb{P}\left[X_v < (1-\varepsilon)\cdot\frac{k}{\delta+1}\right] = \mathbb{P}\left[X_v < (1-\varepsilon)\cdot\mathbb{E}[X_v]\right] < e^{-\varepsilon^2\mathbb{E}[X_v]/2} \leq \frac{1}{N^{\Delta+1}}. \tag{4.1}$$

Algorithm 1 Explicit Deterministic Multicoloring Algorithm: Basic Construction

Input: one-hop view (x, Γ), parameter $\ell \geq 0$
Output: set S of colors, initially $S = \emptyset$
1: **for all** $(\alpha_0, \alpha_1, \ldots, \alpha_\ell) \in \mathbb{F}_{q_0} \times \mathbb{F}_{q_1} \times \cdots \times \mathbb{F}_{q_\ell}$ **do**
2: $\quad \beta_{0,x} := \varphi_{0,x}(\alpha_0)$; $\forall y \in \Gamma : \beta_{0,y} := \varphi_{0,y}(\alpha_0)$
3: $\quad$ **for** $i := 1$ **to** ℓ **do**
4: $\quad\quad \beta_{i,x} := \varphi_{i,\beta_{i-1,x}}(\alpha_i)$; $\forall y \in \Gamma : \beta_{i,y} := \varphi_{i,\beta_{i-1,y}}(\alpha_i)$
5: **if** $\forall y \in \Gamma : \beta_{\ell,x} \neq \beta_{\ell,y}$ **then**
6: $\quad S := S \cup (\alpha_0, \alpha_1, \ldots, \alpha_\ell, \beta_{\ell,x})$

The total number of different possible one-hop views can be bounded as $|\mathcal{N}_1(N, \Delta)| = N \cdot \sum_{\delta=1}^{\Delta} \binom{N-1}{\delta} < N^{\Delta+1}$. By a union bound argument, we therefore get that with positive probability, for all $\delta \in [\Delta]$, all possible one-hop views (x_v, Γ_v) with $|\Gamma_v| = \delta$ get at least $(1-\varepsilon) \cdot k/(\delta+1)$ colors. Hence, there exists a set of k global orders on the ID set $[N]$ such that all one-hop views obtain at least the required number of colors. ∎

Remark: Note that if we increase the number of permutations (i.e., the number of colors) by a constant factor, all possible one-hop views (x, Γ) with $|\Gamma| = \delta$ get a $(1-\varepsilon)/(\delta+1)$-fraction of all colors w.h.p.

4.2. Randomized Algorithms

We will now show that with the use of randomization, the upper bound of Section 4.1 can be significantly improved if the algorithm only needs to be correct w.h.p. We will again use random permutations. The problem of the deterministic algorithm is that the algorithm needs to assign a large set of colors to all roughly N^Δ possible one-hop views. With the use of randomization, we essentially only have to assign colors to n randomly chosen one-hop views.

For simplicity, we assume that every node knows the number of nodes n (knowing an upper bound on n is sufficient). For an integer parameter $k > 0$, every $v \in V$ chooses k independent random numbers $x_{v,1}, \ldots, x_{v,k} \in [kn^4]$ and sends these random numbers to all neighbors. We use these random numbers to induce k random permutations on the nodes. Let $\Gamma(v)$ be the set of neighbors of a node v. A node v selects all colors i for which $x_{v,i} < x_{u,i}$ for all $u \in \Gamma(v)$.

Theorem 4.2. *Choosing $k = 6(\Delta+1)\ln(n)/\varepsilon^2$ leads to a randomized one-shot algorithm that computes a $(1-\varepsilon, k)$-multicoloring w.h.p.*

Remark: In the above algorithm, every node has to generate $\mathcal{O}(\Delta \log^2(n)/\varepsilon^2)$ random bits and send these bits to the neighbors. Using a (non-trivial) probabilistic argument, it is possible to show that the same result can be achieved using only $\mathcal{O}(\log n)$ random bits per node.

4.3. Explicit Algorithms

We have shown in Section 4.1 that there is a deterministic one-shot algorithm that almost matches the lower bound (cf. Theorem 5.2). Unfortunately, the techniques of Section 4.1 do not yield an explicit algorithm. In this section, we will present constructive methods to obtain a one-shot multicoloring algorithm.

We develop the algorithm in two steps. First, we construct a multicoloring where in the worst case, every node v obtains the same fraction of colors independent of v's degree. We then show how to increase the fraction of colors assigned to low-degree nodes. For an integer parameter $\ell \geq 0$,

let $q_0, \dots, q_\ell$ be prime powers and let $d_0, \dots, d_\ell$ be positive integers such that $q_0^{d_0+1} \geq N$ and $q_i^{d_i+1} \geq q_{i-1}$ for $i \geq 1$. For a prime power q and a positive integer d, let $\mathcal{P}(q,d)$ be the set of all q^{d+1} polynomials of degree at most d in $\mathbb{F}_q[z]$, where $\mathbb{F}_q$ is the finite field of order q. We assume that that we are given an injection φ_0 from the ID set $[N]$ to the polynomials in $\mathcal{P}(q_0,d_0)$ and injections φ_i from $\mathbb{F}_{q_{i-1}}$ to $\mathcal{P}(q_i,d_i)$ for $i \geq 1$. For a value x in the respective domain, let $\varphi_{i,x}$ be the polynomial assigned to x by injection φ_i. The first part of the algorithm is an adaptation of a technique used in a coloring algorithm described in [16] that is based on an algebraic construction of [7]. There, a node v with one-hop view (x,Γ) selects a color $\big(\alpha, \varphi_{0,x}(\alpha)\big)$, where $\alpha \in \mathbb{F}_{q_0}$ is a value for which $\varphi_{0,x}(\alpha) \neq \varphi_{0,y}(\alpha)$ for all $y \in \Gamma$ (we have to set q_0 and d_0 such that this is always possible). We make two modifications to this basic algorithm. Instead of only selecting one value $\alpha \in \mathbb{F}_{q_0}$ such that $\forall y \in \Gamma : \varphi_{0,x}(\alpha) \neq \varphi_{0,y}(\alpha)$, we select all values α for which this is true. We then use these values recursively (as if $\varphi_{i,x}(\alpha_i)$ was the ID of v) ℓ times to reduce the dependence of the approximation ratio of the coloring on N. The details of the first step of the algorithm are given by Algorithm 1.

Lemma 4.3. *Assume that for $0 \leq i \leq \ell$, $q_i \geq f_i \Delta d_i$ where $f_i > 1$. Then, Algorithm 1 constructs a multicoloring with $q_\ell \cdot \prod_{i=0}^{\ell} q_i$ colors where every node at least receives a λ/q_ℓ-fraction of all colors where $\lambda = \prod_{i=0}^{\ell}(1-1/f_i)$.*

Proof. All colors that are added to the color set in line 6 are from $\mathbb{F}_{q_0} \times \mathbb{F}_{q_1} \times \cdots \times \mathbb{F}_{q_\ell} \times \mathbb{F}_{q_\ell}$. It is therefore clear that the number of different colors is $q_\ell \cdot \prod_{i=0}^{\ell} q_i$ as claimed. From the condition in line 5, it also follows that the color sets of adjacent nodes are disjoint.

To determine the approximation ratio, we count the number of colors, a node v with one-hop view (x,Γ) gets. First note that the condition in line 5 of the algorithm implies that (and is therefore equivalent to demand that) $\beta_{i,x} \neq \beta_{i,y}$ for all $y \in \Gamma$ and for all $i \in \{0,\dots,\ell\}$ because $\beta_{i,x} = \beta_{i,y}$ implies $\beta_{j,x} = \beta_{j,y}$ for all $j \geq i$. We therefore need to count the number of $(\alpha_0,\dots,\alpha_\ell) \in \mathbb{F}_{q_0} \times \cdots \times \mathbb{F}_{q_\ell}$ for which $\beta_{i,x} \neq \beta_{i,y}$ for all $i \in \{0,\dots,\ell\}$ and all $y \in \Gamma$. We prove by induction on i that for $i < \ell$, there are at least $\prod_{j=0}^{i} q_j \cdot (1-1/f_j)$ tuples $(\alpha_0,\dots,\alpha_i) \in \mathbb{F}_{q_0} \times \cdots \mathbb{F}_{q_i}$ with $\beta_{j,x} \neq \beta_{j,y}$ for all $j \leq i$. Let us first prove the statement for $i = 0$. Because the IDs of adjacent nodes are different, we know that $\varphi_{0,x} \neq \varphi_{0,y}$ for all $y \in \Gamma$. Two different degree d_0 polynomials can be equal at at most d_0 values. Hence, for every $y \in \Gamma$, $\varphi_{0,x}(\alpha) = \varphi_{0,y}(\alpha)$ for at most d_0 values α. Thus, since $|\Gamma| \leq \Delta$, there are at least $q_0 - \Delta d_0 \geq q_0 \cdot (1-1/f_0)$ values α for which $\varphi_{0,x} \neq \varphi_{0,y}$ for all $y \in \Gamma$. This establishes the statement for $i = 0$. For $i > 0$, the argument is analogous. Let $(\alpha_0,\dots,\alpha_{i-1}) \in \mathbb{F}_{q_0} \times \cdots \times \mathbb{F}_{q_{i-1}}$ be such that $\beta_{j,x} \neq \beta_{j,y}$ for all $y \in \Gamma$ and all $j < i$. Because $\beta_{i-1,x} \neq \beta_{i-1,y}$, we have $\varphi_{i,x} \neq \varphi_{i,y}$. Thus, with the same argument as for $i = 0$, there are at least $q_i \cdot (1-1/f_i)$ values α_i such that $\beta_{i,x} \neq \beta_{i,y}$ for all $y \in \Gamma$. Therefore, the number of colors in the color set of every node is at least $\prod_{i=0}^{\ell} q_i \cdot \big(1-1/f_i\big) = \lambda \cdot \prod_{i=0}^{\ell} q_i$. This is a (λ/q_ℓ)-fraction of all colors. ∎

The next lemma specifies how the values of q_i, d_i, and f_i can be chosen to obtain an efficient algorithm.

Lemma 4.4. *Let ℓ be such that $\ln^{(\ell)} N > \max\{e,\Delta\}$. For $0 \leq i \leq \ell$, we can then choose q_i, d_i, and f_i such that Algorithm 1 computes a multicoloring with $\mathcal{O}(\ell\Delta)^{\ell+2} \cdot \log_\Delta N \cdot \log_\Delta \ln^{(\ell)} N$ colors and such that every node gets at least a $1/\big(4e^{9/4}\Delta\lceil \log_\Delta \ln^{(\ell)} N\rceil\big)$-fraction of all colors.*

The number of colors that Algorithm 1 assigns to nodes with degree almost Δ is close to optimal even for small values of ℓ. If we choose $\ell = \Theta(\log^* N - \log^* \Delta)$, nodes of degree $\Theta(\Delta)$ even receive at least a (d/Δ)-fraction of all colors for some constant d. Because the number of colors assigned to a node v is independent of v's degree, however, the coloring of Algorithm 1 is far

Algorithm 2 Explicit Deterministic Multicoloring Algorithm: Small Number of Colors

Input: one-hop view (x, Γ), instances $\mathcal{A}_{2^i,N}$ for $i \in \left[\lceil \log \Delta \rceil\right]$ of Algorithm 1, parameter $\varepsilon \in [0,1]$
Output: set S of colors, initially $S = \emptyset$

1: **for all** $i \in \left[\lceil \log \Delta \rceil\right]$ **do**
2: $\quad \omega_i := \left\lceil \left(\Delta/2^{i-1}\right)^{\varepsilon} \cdot \left|\mathcal{C}_{2^{\lceil \log \Delta \rceil},N}\right| / \left|\mathcal{C}_{2^i,N}\right| \right\rceil$
3: **for all** $i \in \left\{\lceil \log |\Gamma| \rceil, \ldots, \lceil \log \Delta \rceil\right\}$ **do**
4: $\quad$ **for all** $c \in \mathcal{C}_{2^i,N}[x, \Gamma]$ **do**
5: $\quad\quad$ **for all** $j \in [\omega_i]$ **do** $S := S \cup (c, i, j)$

from optimal for low-degree nodes. In the following, we show how to improve the algorithm in this respect.

Let $\mathcal{A}_{\Delta,N}$ be an instance of Algorithm 1 for nodes with degree at most Δ and let $\mathcal{C}_{\Delta,N}$ be the color set of $\mathcal{A}_{\Delta,N}$. Further, for a one-hop view (x, Γ), let $\mathcal{C}_{\Delta,N}[x, \Gamma]$ be the colors assigned to (x, Γ) by Algorithm $\mathcal{A}_{\Delta,N}$. We run instances $\mathcal{A}_{2^i,N}$ for all $i \in \left[\lceil \log \Delta \rceil\right]$. A node v with degree δ chooses the colors of all instances for which $2^i \geq \delta$. In order to achieve the desired trade-offs, we introduce an integer weight ω for each color c, i.e., instead of adding color c, we add colors $(1, c), \ldots, (\omega, c)$. The details are given by Algorithm 2. The properties of Algorithm 2 are summarized by the next theorem. The straight-forward proof is omitted.

Theorem 4.5. *Assume that in the instances of Algorithm 1, the parameter ℓ is chosen such that for all Δ, $\mathcal{A}_{\Delta,N}$ assigns at least a $f(N)/\Delta$-fraction of the colors to every node. Then, for a parameter $\varepsilon \in [0,1]$, Algorithm 2 computes a $\left(\Omega(f(N)\varepsilon/\delta^{\varepsilon}), \mathcal{O}(|\mathcal{C}_{2\Delta,N}| \cdot \Delta^{\varepsilon}/\varepsilon)\right)$-multicoloring.*

Corollary 4.6. *Let $\varepsilon \in [0,1]$ and $\ell \geq 0$ be a fixed constant in all used instances of Algorithm 1. Then, Algorithm 2 computes an $\left(\varepsilon/\mathcal{O}(\delta^{\varepsilon} \log_{\Delta} \ln^{(\ell)} N), \mathcal{O}(\Delta^{\ell+2} \cdot \log_{\Delta} N \cdot \log_{\Delta} \ln^{(\ell)} N)\right)$-multicoloring. In particular, choosing $\ell = 0$ leads to an $\left(\varepsilon/\mathcal{O}(\delta^{\varepsilon} \log_{\Delta} N), \mathcal{O}(\Delta^2 \log_{\Delta}^2 N)\right)$-multicoloring. Taking the maximum possible value for ℓ in all used instances of Algorithm 1 yields an $\left(\varepsilon/\mathcal{O}(\delta^{\varepsilon}), \Delta^{\mathcal{O}(\log^* N - \log^* \Delta)} \cdot \log_{\Delta} N\right)$-multicoloring.*

5. Lower Bounds

In this section, we give lower bounds on the number of colors required for one-shot multicoloring algorithms. In fact, we even derive the lower bounds for algorithms that need to assign only one color to every node, i.e., the results even hold for standard coloring algorithms.

It has been shown in [15] that every deterministic one-shot c-coloring algorithm $\mathcal{A}$ can be interpreted as a set of c antisymmetric relations on the ID set $[N]$. Assume that $\mathcal{A}$ assigns a color from a set C with $|C| = c$ to every one-hop view (x, Γ). For every color $\alpha \in C$, there is a relation $\lhd_{\alpha}$ such that for all $x, y \in [N]$ $x \not\lhd_{\alpha} y \vee y \not\lhd_{\alpha} x$. Algorithm $\mathcal{A}$ can assign color $\alpha \in C$ to a one-hop view (x, Γ) iff $\forall y \in \Gamma : x \lhd_{\alpha} y$.

For $\alpha \in C$, let $\mathrm{Bad}_{\alpha}(x) := \{y \in [N] : x \not\lhd_{\alpha} y\}$ be the set of IDs that must not be adjacent to an α-colored node with ID x. To show that there is no deterministic, one-shot c-coloring algorithm, we need to show that for every c antisymmetric relations $\lhd_{\alpha_1}, \ldots, \lhd_{\alpha_c}$ on $[N]$, there is a one-hop view (x, Γ) such that $\forall i \in [c] : \Gamma \cap \mathrm{Bad}_{\alpha_i}(x) \neq \emptyset$. The following lemma is a generalization of Lemma 4.5 in [15] and key for the deterministic and the randomized lower bounds. As the proof is along the same lines as the proof of Lemma 4.5 in [15], it is omitted here.

Lemma 5.1. *Let $X \subseteq [N]$ be a set of IDs and let $t_1, \dots, t_\ell$ and $k_1, \dots, k_\ell$ be positive integers such that*

$$t_i \cdot \big(\lambda(|X| - c)t_i - c\big) > 2c(k_i - 1) \quad \text{for } 1 \leq i \leq \ell \text{ and a parameter } \lambda \in [0,1].$$

Then there exists an ID set $X' \subseteq X$ with $|X'| > (1 - \ell \cdot \lambda) \cdot (|X| - c)$ such that for all $i \in [\ell]$,

$$\forall x \in X', \forall \alpha_1, \dots, \alpha_{t_i} \in C : \sum_{j=1}^{t_i} \big|\mathrm{Bad}_{\alpha_j}(x) \cap X\big| \geq k_i, \quad \forall x \in X', \forall \alpha \in C : \mathrm{Bad}_\alpha(x) \cap X \neq \emptyset.$$

Based on several applications of Lemma 5.1 (and based on an $\Omega(\log\log N)$ lower bound in [16]), it is possible to derive an almost tight lower bound for deterministic one-shot coloring algorithms. Due to lack of space, we only state the result here.

Theorem 5.2. *If $N = \Omega(\Delta^2 \log \Delta)$, every deterministic one-shot coloring algorithm needs at least $\Omega(\Delta^2 + \log\log N)$ colors.*

5.1. Randomized Lower Bound

To obtain a lower bound for randomized multicoloring algorithms, we can again use the tools derived for the deterministic lower bound by applying Yao's principle. On a worst-case input, the best randomized algorithm cannot perform better than the best deterministic algorithm for a given random input distribution. Choosing the node labeling at random allows to again only consider deterministic algorithms.

We assume that the n nodes are assigned a random permutation of the labels $1, \dots, n$ (i.e., every label occurs exactly once). Note that because we want to prove a lower bound, assuming the most restricted possible ID space makes the bound stronger. For an ID $x \in [n]$, we sort all colors $\alpha \in C$ by increasing values of $|\mathrm{Bad}_\alpha(x)|$ and let $\alpha_{x,i}$ be the i^{th} color in this sorted order. Further, for $x \in [n]$, we define $b_{x,i} := \big|\mathrm{Bad}_{\alpha_{x,i}}(x)\big|$. In the following, we assume that

$$c = \kappa \cdot \frac{\Delta \lfloor \ln n \rfloor}{\lceil \ln \ln n \rceil + 2} \quad \text{and} \quad n \geq 12 \quad \text{and} \quad n \geq \Delta \cdot \ln n \tag{5.1}$$

for a constant $0 < \kappa \leq 1$ that will be determined later. By applying Lemma 5.1 in different ways, the next lemma gives lower bounds on the values of $b_{x,i}$ for $n/2$ IDs $x \in [n]$.

Lemma 5.3. *Assume that c and n are as given by Equation* (5.1) *and let $0 < \rho < 1/3$ be a positive constant. Further, let $\tilde{t} = \big\lceil \rho \ln n / \ln\ln n \big\rceil$ and $t_i = 2^{i-1} \cdot \lfloor \ln n \rfloor$ for $1 \leq i \leq \ell$ where $\ell = \lceil \ln \ln n \rceil + 2$. Then, for at least $n/2$ of all IDs $x \in [n]$, we have*

$$b_{x,1} \geq \frac{\ln\ln n}{44\kappa \cdot \ln n} \cdot \frac{n}{\Delta} - 1, \quad b_{x,\tilde{t}} \geq \frac{\rho}{48\kappa} \cdot \frac{n}{\Delta} - \frac{1}{2}, \quad b_{x,t_i} \geq 2^{i-1} \cdot \left(\frac{1}{8\kappa} \cdot \frac{n}{\Delta} - \frac{1}{2}\right) \quad \text{for } 1 \leq i \leq \ell.$$

In order to prove the lower bound, we want to show that for a randomly chosen one-hop view (x, Γ) with $|\Gamma| = \Delta$, the probability that there is a color $\alpha \in C$ for which $\Gamma \cap \mathrm{Bad}_\alpha(x) = \emptyset$ is sufficiently small. Instead of directly looking at random one-hop views (x, Γ) with $|\Gamma| = \Delta$, we first look at one-hop views with $|\Gamma| \approx \Delta/e$ that are constructed as follows. Let $X \subseteq [n]$ be the set of IDs x of size $|X| \geq n/2$ for which the bounds of Lemma 5.3 hold. We choose x_R uniformly at random from X. The remaining $n - 1$ IDs are independently added to a set Γ_R with probability $p = \frac{\Delta}{en}$. For a color $\alpha \in C$, let $\mathcal{E}_\alpha$ be the event that $\Gamma_R \cap \mathrm{Bad}_\alpha(x_R) \neq \emptyset$, i.e., $\mathcal{E}_\alpha$ is the event that color α cannot be assigend to the randomly chosen one-hop view (x_R, Γ_R).

Lemma 5.4. *The probability that the randomly chosen one-hop view cannot be assigned one of the c colors in C is bounded by*

$$\mathbb{P}\left[\bigcap_{\alpha\in C}\mathcal{E}_\alpha\right] \;\geq\; \prod_{\alpha\in C}\mathbb{P}\big[\mathcal{E}_\alpha\big] \;\geq\; \prod_{\alpha\in C}\left(1-e^{-\frac{\Delta}{en}\cdot|\mathrm{Bad}_\alpha(x_R)|}\right) \;=\; \prod_{i=1}^{c}\left(1-e^{-\frac{\Delta\cdot b_{x_R,i}}{en}}\right).$$

Proof. Note first that for $\alpha\in C$, we have

$$\mathbb{P}\big[\overline{\mathcal{E}_\alpha}\big] \;=\; \mathbb{P}\big[\Gamma_R\cap\mathrm{Bad}_\alpha(x_R)=\emptyset\big] \;=\; (1-p)^{|\mathrm{Bad}_\alpha(x_R)|} \;\leq\; e^{-p|\mathrm{Bad}_\alpha(x_R)|} \;=\; e^{-\frac{\Delta}{en}\cdot|\mathrm{Bad}_\alpha(x_R)|}.$$

It therefore remains to prove that the probability that all events $\mathcal{E}_\alpha$ occur can be lower bounded by the probability that would result for independent events. Let us denote the colors in C by $\alpha_1,\ldots,\alpha_c$. We then have

$$\mathbb{P}\left[\bigcap_{\alpha\in C}\mathcal{E}_\alpha\right] \;=\; \prod_{i=1}^{c}\mathbb{P}\left[\mathcal{E}_{\alpha_i}\middle|\bigcap_{j=1}^{i-1}\mathcal{E}_{\alpha_j}\right] \;\geq\; \prod_{i=1}^{c}\mathbb{P}\big[\mathcal{E}_{\alpha_i}\big]. \tag{5.2}$$

The inequality holds because the events $\mathcal{E}_\alpha$ are positively correlated. Knowing that an element from a set $\mathrm{Bad}_\alpha(x_R)$ is in Γ_R cannot decrease the probability that an element from a set $\mathrm{Bad}_{\alpha'}(x_R)$ is in Γ_R. Note that this is only true because the IDs are independently added to Γ_R. More formally, Inequality (5.2) can also directly be followed from the FKG inequality [8]. ∎

For space reasons, the following two lemmas are given without proof.

Lemma 5.5. *Assume that c and n are given as in* (5.1) *where the constant κ is chosen sufficiently small and let $\rho>0$ be a constant as in Lemma 5.3. There is a constant $n_0>0$ such that for $n\geq n_0$, $\mathbb{P}\left[\bigcap_{\alpha\in C}\mathcal{E}_\alpha\right] > \frac{1}{2n^{3\rho}}$.*

Lemma 5.6. *Let (x,Γ) be a one-hop view chosen uniformly at random from all one-hop views with $|\Gamma|=\Delta$. If $\Delta\geq e(\ln n+2)$ and n, c, and ρ are as before, the probability that none of the c colors can be assigned to (x,Γ) is at least $1/(8n^{3\rho})$.*

In the following, we call a node u together with Δ neighbors $v_1,\ldots,v_\Delta$, a Δ-star.

Theorem 5.7. *Let G be a graph with n nodes and $2n^\varepsilon$ disjoint Δ-stars for a constant $\varepsilon>0$. On G, every randomized one-shot coloring algorithm needs at least $\Omega(\Delta\log n/\log\log n)$ colors in expectation and with high probability.*

Proof. W.l.o.g., we can certainly assume that $n\geq n_0$ for a sufficiently large constant n_0. We choose $\rho\leq\varepsilon/4$ and consider n^ε of the $2n^\varepsilon$ disjoint Δ-stars. Let us call these n^ε Δ-stars $S_1,\ldots,S_{n^\varepsilon}$. Assume that the ID assignment of the n nodes of G is chosen uniformly at random from all ID assignments with IDs $1,\ldots,n$. The IDs of the star S_1 are perfectly random. We can therefore directly apply Lemma 5.6 and obtain that the probability that the center node of S_1 gets no color is at least $1/(8n^{3\rho})$. Consider star S_2. The IDs of the nodes of S_2 are chosen at random among the $n-\Delta-1$ IDs that are not assigned to the nodes of S_1. Applying Lemma 5.6 we get that the probability that S_2 does not get a color is at least $1/(8(n-\Delta-1)^{3\rho})\geq 1/(8n^{3\rho})$ independently of whether S_1 does get a color. The probability that the starts $S_1,\ldots,S_{n^\varepsilon}$ all get a color therefore is at most

$$\prod_{i=0}^{n^\varepsilon-1}\left(1-\frac{1}{8(n-i(\Delta+1))^{3\rho}}\right)\leq\left(1-\frac{1}{8n^{3\rho}}\right)^{n^\varepsilon}\leq e^{-\frac{n^\varepsilon}{8n^{3\rho}}}\leq e^{-n^\rho/8}.$$

Hence, there is a constant $\eta>0$ such that $\eta\Delta\ln n/\ln\ln n$ colors do not suffice with probability at least $1-e^{-n^\rho/8}$ for a positive constant ρ. The lemma thus follows. ∎

References

[1] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *Proc. of 30th Symposium on Foundations of Computer Science (FOCS)*, pages 364–369, 1989.

[2] B. Balasundaram and S. Butenko. Graph domination, coloring and cliques in telecommunications. In M. Resende and P. Pardalos, editors, *Handbook of Optimization in Telecommunications*, pages 865–890. Springer, 2006.

[3] L. Barenboim and M. Elkin. Sublogarithmic distributed mis algorithm for sparse graphs using nash-williams decomposition. In *Proc. of 27th ACM Symposium on Principles of Distributed Computing (PODC)*, 2008.

[4] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.

[5] A. Czygrinow, M. Hańćkowiak, and M. Karoński. Distributed $\mathrm{O}(\delta \log n)$-edge-coloring algorithm. In *Proc. of 9th Annual European Symposium on Algorithms (ESA)*, volume 2161 of *LNCS*, pages 345–355, 2001.

[6] G. De Marco and A. Pelc. Fast distributed graph coloring with $\mathrm{O}(\Delta)$ colors. In *Proc. of 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 630–635, 2001.

[7] P. Erdős, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel Journal of Mathematics*, 51:79–89, 1985.

[8] C. Fortuin, J. Ginibre, and P. Kasteleyn. Correlation inequalities on some partially ordered sets. *Comm. in Mathematical Physics*, 22:89–103, 1971.

[9] P. Fraigniaud, C. Gavoille, D. Ilcinkas, and A. Pelc. Distributed computing with advice: Information sensitivity of graph coloring. In *Proc. of 34th Int. Coll. on Automata, Languages and Programming*, 2007.

[10] S. Gandham, M. Dawande, and R. Prakash. Link scheduling in sensor networks: Distributed edge coloring revisited. In *Proc. of 24th IEEE Conference on Computer Communications (INFOCOM)*, pages 2492–2501, 2005.

[11] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434–446, 1988.

[12] D. A. Grable and A. Panconesi. Nearly optimal distributed edge coloring in $\mathrm{O}(\log \log n)$ rounds. *Random Structures and Algorithms*, 10(3):385–405, 1997.

[13] T. Herman and S. Tixeuil. A distributed TDMA slot assignment algorithm for wireless sensor networks. In *Proc. of 1st Int. Workshop on Algorithmic Aspects of Wireless Sensor Networks*, pages 45–58, 2004.

[14] K. Kothapalli, M. Onus, C. Scheideler, and C. Schindelhauer. Distributed coloring in $O(\sqrt{\log n})$ bit rounds. In *Proc. of 20th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[15] F. Kuhn and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proc. of 25th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 7–15, 2006.

[16] N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.

[17] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036–1053, 1986.

[18] S. Mecke. MAC layer and coloring. In D. Wagner and R. Wattenhofer, editors, *Algorithms for Sensor and Ad Hoc Networks*, pages 63–80, 2007.

[19] T. Moscibroda and R. Wattenhofer. Coloring unstructured radio networks. In *Proc. of 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 39–48, 2005.

[20] M. Naor and L. Stockmeyer. What can be computed locally? In *Proc. of 25th ACM Symposium on Theory of Computing (STOC)*, pages 184–193, 1993.

[21] A. Panconesi and A. Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):581–592, 1995.

[22] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the chernoff-hoeffding bounds. *SIAM Journal on Computing*, 26(2):350–368, 1997.

[23] A. Pelc. Personal communication.

[24] S. Ramanathan. A unified framework and algorithm for channel assignment in wireless networks. *Wireless Networks*, 5:81–94, 1999.

[25] I. Rhee, A. Warrier, J. Min, and L. Xu. DRAND: Distributed randomized TDMA scheduling for wireless ad-hoc networks. In *7th ACM Symp. on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 190–201, 2006.

[26] J. Schneider and R. Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proc. of 27th ACM Symposium on Principles of Distributed Computing (PODC)*, 2008.

[27] X. Zhang, J. Hong, L. Zhang, X. Shan, and V. Li. CP-TDMA: Coloring- and probability-based TDMA scheduling for wireless ad hoc networks. *IEICE Transactions on Communication*, E91-B(1):322–326, 2008.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 625–636
www.stacs-conf.org

EFFICIENT ISOMORPHISM TESTING FOR A CLASS OF GROUP EXTENSIONS

FRANÇOIS LE GALL

ERATO-SORST Quantum Computation and Information Project,
Japan Science and Technology Agency, Tokyo
E-mail address: legall@qci.jst.go.jp

Abstract. The group isomorphism problem asks whether two given groups are isomorphic or not. Whereas the case where both groups are abelian is well understood and can be solved efficiently, very little is known about the complexity of isomorphism testing for nonabelian groups. In this paper we study this problem for a class of groups corresponding to one of the simplest ways of constructing nonabelian groups from abelian groups: the groups that are extensions of an abelian group A by a cyclic group $\mathbb{Z}_m$. We present an efficient algorithm solving the group isomorphism problem for all the groups of this class such that the order of A is coprime with m. More precisely, our algorithm runs in time almost linear in the orders of the input groups and works in the general setting where the groups are given as black-boxes.

1. Introduction

The group isomorphism problem is the problem of deciding, for two given groups G and H, whether there exists an isomorphism between G and H, i.e. a one-one map preserving the group operation. This is a fundamental problem in computational group theory but little is known about its complexity. It is known that the group isomorphism problem (for groups given by their multiplication tables) reduces to the graph isomorphism problem [12], and thus the group isomorphism problem is in the complexity class $NP \cap coAM$ (since the graph isomorphism problem is in this class [2]). Miller [14] has developed a general technique to check group isomorphism in time $O(n^{\log n + O(1)})$, where n denotes the size of the input groups and Lipton, Snyder and Zalcstein [13] have given an algorithm working in $O(\log^2 n)$ space. However, no polynomial algorithm is known for the general case of this problem.

Another line of research is the design of algorithms solving the group isomorphism problem for particular classes of groups. For abelian groups polynomial time algorithms follow directly from efficient algorithms for the computation of Smith normal form of integer matrices [10, 6]. More efficient methods have been given by Vikas [22] and Kavitha [11] for groups given by their multiplication tables. The current fastest algorithm solving the abelian group isomorphism problem for groups given as black-boxes has been developed

1998 ACM Subject Classification: F.2.2 Nonnumerical Algorithms and Problems.
Key words and phrases: polynomial-time algorithms, group isomorphism, black-box groups.

by Buchmann and Schmidt [5] and works in time $O(n^{1/2}(\log n)^{O(1)})$. However, as far as nonabelian groups are concerned, very little is known. For solvable groups Arvind and Torán [1] have shown that the group isomorphism problem is in $NP \cap coNP$ under certain complexity assumptions but, to our knowledge, the only polynomial-time algorithm testing isomorphism of a nontrivial class of nonabelian groups is a result by Garzon and Zalcstein [7], and holds for a very restricted class.

In this work we focus on the worst-case complexity of the group isomorphism problem over classes of nonabelian groups. Since for abelian groups the problem can be solved efficiently, we study one of the most natural next targets: cyclic extensions of abelian groups. Loosely speaking such extensions are constructed by taking an abelian group A and adding one element y that, in general, does not commute with the elements in A. More formally the class of groups we consider in this paper, denoted $\mathscr{S}$, is the following.

Definition 1.1. Let G be a finite group. We say that G is in the class $\mathscr{S}$ if there exists a normal abelian subgroup A in G and an element $y \in G$ of order coprime with $|A|$ such that $G = \langle A, y \rangle$.

In technical words G is an extension of an abelian group A by a cyclic group $\mathbb{Z}_m$ with $gcd(|A|, m) = 1$. This class of groups includes all the abelian groups and many non-abelian groups too. For example, for $A = \mathbb{Z}_3^4$ and $m = 4$ there are exactly 9 isomorphism classes in $\mathscr{S}$.

A group can be represented on a computer in different ways. In this paper we use the black-box setting introduced by Babai and Szemerédi [4], which is one of the most general models for handling groups, and particularly convenient to discuss algorithms running in sublinear time. In order to state precisely the running time of our algorithm, we introduce the following definition. For any group G in the class $\mathscr{S}$, let $\gamma(G)$ be the smallest integer m such that G is an extension of an abelian group A by the cyclic group $\mathbb{Z}_m$ with $gcd(|A|, m) = 1$. The main result of this paper is the following theorem.

Theorem 1.2. *There exists a deterministic algorithm checking whether two groups G and H in the class $\mathscr{S}$ given as black-box groups are isomorphic and, if this is the case, computing an isomorphism from G to H. Its running time has for upper bound $(\sqrt{n} + \gamma)^{1+o(1)}$, where $n = min(|G|, |H|)$ and $\gamma = min(\gamma(G), \gamma(H))$.*

Notice that, for any group G in the class $\mathscr{S}$, the relation $\gamma(G) \leq |G|$ holds. Then the complexity of our algorithm has for upper bound $n^{1+o(1)}$, and is almost linear in the size of the groups. Another observation is that, if $\gamma = O(n^{1/2})$, then the complexity of our algorithm is $n^{1/2+o(1)}$ and is of the same order as the best known algorithm testing isomorphism of abelian groups [5] in the black-box setting. This case $\gamma = O(n^{1/2})$ corresponds to the rather natural problem of testing isomorphism of extensions of a large abelian group by a small cyclic group.

The outline of our algorithm is as follows. Since a group G in the class $\mathscr{S}$ may in general be written as the extension of an abelian group A_1 by a cyclic group $\mathbb{Z}_{m_1}$ and as the extension of an abelian group A_2 by a cyclic group $\mathbb{Z}_{m_2}$ with $A_1 \ncong A_2$ and $m_1 \neq m_2$, we introduce (in Section 3) the concept of a standard decomposition of G, which is an invariant for the groups in the class $\mathscr{S}$ in the sense that two isomorphic groups have similar standard decompositions (but the converse is false). We also show how to compute a standard decomposition of G efficiently. This allows us to consider only the case where H and G are two extensions of the same abelian group A by the same cyclic group $\mathbb{Z}_m$. One of the

main technical contributions of this paper is an efficient algorithm that tests whether two automorphisms of order m in the automorphism group of A are conjugate or not (Section 4). Finally, we present a time-efficient reduction from the problem of testing whether G and H are isomorphic to an instance of the above conjugacy problem (Section 5).

Remark. Several algorithms for the group isomorphism problem performing relatively well in practice are known and have been implemented in computational group theory softwares (GAP, MAGMA,...). The main works in this area are the algorithms developed by Smith for solvable groups [20] and by O'Brien [15] for p-groups. However these algorithms involve computation in groups of size exponential in n, e.g. the automorphism groups or the cohomology groups, and no rigorous analysis of their time complexity is available.

2. Preliminaries

We assume that the reader is familiar with the basic notions of group theory and state without proofs basic definitions and properties of groups we will use in this paper.

Let G be a finite group (in this paper we will consider only finite groups). Given a set S of elements of G, the subgroup generated by the elements of G is written $\langle S\rangle$. For any two elements $g, h \in G$ we denote $[g, h]$ the commutator of g and h, i.e. $[g, h] = ghg^{-1}h^{-1}$. The commutator subgroup of G is defined as $G' = \langle [g, h] \,|\, g, h \in G\rangle$. The derived series of G is defined recursively as $G^{(0)} = G$ and $G^{(i+1)} = (G^{(i)})'$. The group G is said to be solvable if there exists some integer k such that $G^{(k)} = \{e\}$.

Given a prime p, a p-group is a group of order p^r for some integer r. It is well-known that any p-group is solvable. If G is a group and $|G| = p_1^{e_i} \dots p_r^{e_r}$ for distinct prime numbers p_i such that $p_1 < \dots < p_r$, then for each $i \in \{1, \dots, r\}$ the group G has a subgroup of order $p_i^{e_i}$ called a Sylow p_i-subgroup of G. Moreover, if G is additionally abelian, then each Sylow p_i-group is unique and G is the direct product of its Sylow subgroups. Abelian p-groups have remarkably simple structures: any abelian p-group P is isomorphic to a direct product of cyclic p-groups $\mathbb{Z}_{p^{e_1}} \times \dots \times \mathbb{Z}_{p^{e_s}}$ for some positive integer s and positive integers $e_1 \leq \dots \leq e_s$, and this decomposition is unique. A total order $\preceq$ over the set of prime powers can be defined as follows: for any two prime powers p^α and q^β where α and β are positive integers, we write $p^\alpha \preceq q^\beta$ if and only if $(p < q)$ or $(p = q$ and $\alpha \leq \beta)$. We say that a list $(g_1, \dots, g_t)$ of t elements in G is a basis of an abelian group G if $G = \langle g_1\rangle \times \dots \times \langle g_t\rangle$, the order of each g_i is a prime power and $|g_i| \preceq |g_j|$ for any $1 \leq i \leq j \leq n$. It is easy to show that any (finite) abelian group has a basis and that, if $(g_1, \dots, g_t)$ and $(g'_1, \dots, g'_{t'})$ are two bases of G, then $t = t'$ and $|g_i| = |g'_i|$ for each $i \in \{1, \dots, t\}$. For example, $(g_1, \dots, g_t)$ is a basis of $G \cong \mathbb{Z}_2 \times \mathbb{Z}_4 \times \mathbb{Z}_3^2$ if and only if $t = 4$, $|g_1| = 2$, $|g_2| = 4$, $|g_3| = |g_4| = 3$ and $G = \langle g_1\rangle \times \langle g_2\rangle \times \langle g_3\rangle \times \langle g_4\rangle$.

Let n be a positive integer. A Hall divisor of n is a positive integer m dividing n such that m is coprime with n/m. A subgroup H of a finite group G is called a Hall subgroup of G if $|H|$ is a Hall divisor of $|G|$. We will use in this paper the following well-known theorem.

Theorem 2.1 (Hall's theorem)**.** *Let G be a finite solvable group and r be a Hall divisor of $|G|$. If H_1 and H_2 are two subgroups of G with $|H_1| = |H_2| = r$, then H_1 and H_2 are conjugate.*

We say that a finite group G is an extension of a group K by a group L if there exists a normal abelian subgroup $N \cong K$ of G such that $G/N \cong L$. We say that such an extension splits if there exists some subgroup M of G such that $G = NM$ and $N \cap M = \{e\}$. The

Schur-Zassenhaus theorem states that any extension of K by L such that $gcd(|K|, |L|) = 1$ splits. Concretely, any such split extensions can be constructed as a semidirect product $K \rtimes L$. Thus an equivalent definition of the class $\mathscr{S}$ is the following: a group G is in $\mathscr{S}$ if and only if there exist an abelian group A and a cyclic group $\mathbb{Z}_m$ with $gcd(|A|, m) = 1$ such that $G = A \rtimes \mathbb{Z}_m$.

In this paper we work in the black-box setting first introduced in [4]. A black-box group is a representation of a group where elements are represented by strings (of the same length). An oracle that performs the group product is available: given two strings representing two elements g and g', the oracle outputs the string representing $g \cdot g'$. Another oracle that, given a string representing an element g, computes a string representing the inverse g^{-1} is available as well. In this paper we assume the usual unique encoding hypothesis, i.e. any element of the group is encoded by a unique string. We say that a group G is input as a black-box if a set of strings representing generators $\{g_1, \ldots, g_s\}$ of G with $s = O(\log |G|)$ is given as input, and queries to the multiplication and inversion oracles can be done at cost 1. The hypothesis on s is natural since every group G has a generating set of size $O(\log |G|)$, and enables us to make the exposition of our results easier. Also notice that a set of generators of any size can be converted efficiently into a set of generators of size $O(\log |G|)$ if randomization is allowed [3].

3. Computing a Standard Decomposition

For a given group G in the class $\mathscr{S}$ in general many different decompositions as a semidirect product of an abelian group by a cyclic group exist. For example, the abelian group $\mathbb{Z}_6 = \langle x_1, x_2 \mid x_1^2 = x_2^3 = [x_1, x_2] = e \rangle$ can be written as $\langle x_1 \rangle \times \langle x_2 \rangle$, $\langle x_2 \rangle \times \langle x_1 \rangle$ or $\langle x_1, x_2 \rangle \times \{e\}$. That is why we introduce the notion of a standard decomposition. Let us first start with a simple definition.

Definition 3.1. Let G be a finite group. For any positive integer m denote by $\mathscr{D}_G^m$ the set (possibly empty) of pairs (A, B) such that the following three conditions hold: (i) A is a normal abelian subgroup of G of order coprime with m; and (ii) B is a cyclic subgroup of G of order m; and (iii) $G = AB$.

Notice that if for some m the set $\mathscr{D}_G^m$ is not empty, then G is in the class $\mathscr{S}$. Conversely, if G is in $\mathscr{S}$, then there exists at least one integer m such that $\mathscr{D}_G^m$ is not empty. Also notice that $\gamma(G)$ is the smallest positive integer such that $\mathscr{D}_G^{\gamma(G)} \neq \varnothing$. We now define the concept of a standard decomposition.

Definition 3.2. Let G be a group in the class $\mathscr{S}$. A standard decomposition of G is an element of $\mathscr{D}_G^{\gamma(G)}$.

Before explaining how to compute a standard definition for a group in $\mathscr{S}$, let us mention that it is well known that the order of an element g of any finite group G can be computed deterministically in time $\tilde{O}(|G|^{1/2})$ using Shanks' baby-step/giant-step method [18] or its variants [19]. In the following proposition we show that the decomposition of an element in an abelian group can be found efficiently by a very similar approach (we will need this in Section 5).

Proposition 3.3. *Let A be an abelian group and $(g_1, \ldots, g_s)$ be a basis of A. There exists a deterministic algorithm with time complexity $\tilde{O}(|A|^{1/2})$ that, given any element $g \in A$, outputs integers $a_1, \ldots, a_s$ such that $g = g_1^{a_1} \cdots g_s^{a_s}$.*

Proof. Denote $r_i = \sqrt{|g_i|}$ for each $i \in \{1, \ldots, s\}$ and, for simplicity, suppose that r_i is an integer. The case where r_i is not an integer is similar. The algorithm first computes the set $S = \{g_1^{c_1} \cdots g_s^{c_s} \mid c_i \in \{0, \ldots, r_i - 1\}\}$. Then the algorithm tries all the elements $(b_1, \ldots, b_s)$ with $b_i \in \{0, \ldots, r_i - 1\}$ until finding an element $(\bar{b}_1, \ldots, \bar{b}_s)$ such that $g g_1^{-\bar{b}_1 r_1} \cdots g_s^{-\bar{b}_s r_s} \in S$. Denote $g g_1^{-\bar{b}_1 r_1} \cdots g_s^{-\bar{b}_s r_s} = g_1^{c_1} \cdots g_s^{c_s}$, where each c_i is an element of $\{1, \ldots, r_i - 1\}$. A clever way for finding the c_i's is to use an appropriate data structure for storing S. Then the algorithm outputs $(r_1\bar{b}_1 + c_1, \ldots, r_s\bar{b}_s + c_s)$. The correctness of this algorithm follows immediately from the fact that, if $g = g_1^{a_1} \cdots g_s^{a_s}$, then each a_i can be written as $a_i = \bar{b}_i r_i + c_i$ for some $\bar{b}_i$ and c_i in $\{0, \ldots, r_i - 1\}$. Its complexity is $\tilde{O}(|A|^{1/2})$. ∎

We now show how to compute a standard decomposition of any group in the class $\mathscr{S}$ in time polynomial in the order of the group. The key part of the algorithm is the following procedure FIND-DECOMPOSITION that, given a group G in $\mathscr{S}$ and an integer m, computes an element of $\mathscr{D}_G^m$ if this set is not empty. The description is given in metacode, followed by more details.

Procedure FIND-DECOMPOSITION(G, m)

INPUT: a set of generators $\{g_1, \ldots, g_s\}$ of a group G in $\mathscr{S}$ with $s = O(\log |G|)$
a positive integer m dividing $|G|$
OUTPUT: an error message or a pair (M, z) where $z \in G$ and M is a subset of G

1 compute a set of generators $\{x_1, \ldots, x_t\}$ of G' with $t = O(\log |G|)$;
2 factorize m and write $m = p_1^{e_1} \cdots p_r^{e_r}$;
3 search indexes $k_1, \ldots, k_r \in \{1, \ldots, s\}$ such that $p_\ell^{e_\ell}$ divides $|g_{k_\ell}|$ for each $1 \le \ell \le r$;
4 **if** no such r-uple $(k_1, \ldots, k_r)$ exists
5 **then** return ERROR;
6 **else**
7 $g \leftarrow \Pi_{\ell=1}^{r} g_{k_\ell}^{|g_{k_\ell}|/p_\ell^{e_\ell}}$;
8 **if** m does not divide $|g|$
9 **then** return ERROR;
10 **else**
11 $z \leftarrow g^{|g|/m}$;
12 **for** $j = 1$ **to** s **do** $h_j \leftarrow g_j^m$;
13 **if** $\langle x_1, \ldots, x_t, h_1, \ldots, h_s \rangle$ is abelian
and $gcd(|x_i|, m) = 1$ for each $i \in \{1, \ldots, t\}$
and $gcd(|h_\ell|, m) = 1$ for each $\ell \in \{1, \ldots, s\}$
14 **then** return $(\{x_1, \ldots, x_t, h_1, \ldots, h_s\}, z)$;
15 **else** return ERROR;
16 **endelse**
17 **endelse**

At Step 1 a set of generators $\{z_1, \ldots, z_{t'}\}$ of G' with $t' = O(s^3)$ can be computed using $O(s^3)$ group operations by noticing that $G' = \langle g_k[g_i, g_j]g_k^{-1} \mid i, j, k \in \{1, \ldots, s\} \rangle$ (we refer to [9] for a proof of this simple fact). Since G' is abelian for any group G in the class $\mathscr{S}$, a generating set $\{x_1, \ldots, x_t\}$ of G' with $t = O(\log |G|)$ can then be obtained in time $\tilde{O}(|G|^{1/2})$ using the deterministic algorithm by Buchmann and Schmidt [5] that computes a basis of any abelian group K in time $\tilde{O}(|K|^{1/2})$. At Step 2 the naive technique for factoring m (trying all the integers up to $\sqrt{m}$) is sufficient. This takes $\tilde{O}(|G|^{1/2})$ time. At Steps 3, 7

and 13 we use Shanks' method [18] to compute orders of elements of G in time $\tilde{O}(|G|^{1/2})$. At step 13, commutativity is tested by checking that every two generators commute: this can be done in $O(s^2+t^2)$ group operations. Proposition 3.6 below summarizes the time complexity of the procedure and prove its correctness. We state first two simple lemmas.

Lemma 3.4. *Let G be a group in $\mathscr{S}$ and m be any positive integer. If (A_1,B_1) and (A_2,B_2) are two elements of $\mathscr{D}_G^m$, then $A_1=A_2$.*

Proof. Let us write $B_1=\langle y_1\rangle$. Any element g of A_2 can be written as $g=hy_1^c$ with $h\in A_1$ and some integer c. If $c\not\equiv 0 \bmod m$, then $gcd(m,|g|)\neq 1$, which is excluded since $|A_2|$ and m are coprime. Then $A_2\subseteq A_1$. By symmetry $A_1\subseteq A_2$ and $A_1=A_2$. ∎

Lemma 3.5. *Let G be a group in $\mathscr{S}$ and (A,B) be a standard decomposition of G. Denote $|B|=m$. Let $\{g_1,\ldots,g_s\}$ be a set of generators of G. Then $A=\langle G', g_1^m,\ldots,g_s^m\rangle$, where G' is the derived subgroup of G.*

Proof. Let $B=\langle y\rangle$ and, for each $i\in\{1,\ldots,s\}$, write g_i as $z_iy^{k_i}$ for some $z_i\in A$ and $k_i\in\{1,\ldots,m\}$. Then $A=\langle G', z_1,\ldots,z_s\rangle$. Notice that G' has to be included since in general $A\neq\langle z_1,\ldots,z_s\rangle$, e.g. $G=\langle x_1,x_2,y \mid x_1^3=x_2^3=y^2=e, yx_1=x_2y, yx_2=x_1y\rangle$ with the generating set $g_1=x_1y$ and $g_2=y$. A simple computation shows that $g_i^m=u_iz_i^my^{mk_i}=u_iz_i^m$ for some element $u_i\in G'$. Since m is coprime with the order of z_i, we conclude that $A=\langle G', g_1^m,\ldots,g_s^m\rangle$. ∎

Proposition 3.6. *The time complexity of the procedure* Find-Decomposition(G,m) *is $\tilde{O}(|G|^{1/2})$. If $\mathscr{D}_G^m\neq\varnothing$, then* Find-Decomposition(G,m) *outputs a pair (M,z) such that $(\langle M\rangle,\langle z\rangle))\in\mathscr{D}_G^m$. Conversely, if* Find-Decomposition(G,m) *does not output an error message, then its output (M,z) is such that $\langle M,z\rangle\in\mathscr{S}$ and $(\langle M\rangle,\langle z\rangle)\in\mathscr{D}^m_{\langle M,z\rangle}$.*

Proof. It is clear that the procedure always terminates since no loop is used. The time complexity follows from the analysis of Steps 1, 2, 3, 7 and 13 already done, and from the fact that $s=O(\log|G|)$.

Suppose that $\mathscr{D}_G^m\neq\varnothing$ and take a decomposition $(A,\langle y\rangle)\in\mathscr{D}_G^m$. Write $m=p_1^{e_1}\cdots p_r^{e_r}$ for primes $p_1<\cdots<p_r$ and denote $q_\ell=p_\ell^{e_\ell}$ for each $\ell\in\{1,\ldots,r\}$. Notice that for any generating set $\{g_1,\ldots,g_s\}$ of G, and for each $\ell\in\{1,\ldots,r\}$, there should be some index k_ℓ for which g_{k_ℓ} is of the form $u_\ell y^{c_\ell}$, where $u_\ell\in A$ and c_ℓ is such that q_ℓ divides the order of y^{c_ℓ}, i.e. q_ℓ divides $m/gcd(m,c_\ell)$. Also notice that in this case q_ℓ divides the order of g_{k_ℓ} as well. Then the element $\bar{g}_{k_\ell}=g_{k_\ell}^{|g_{k_\ell}|/q_\ell}$ has order q_ℓ and, more precisely, is of the form $v_\ell y^{d_\ell}$ for some $v_\ell\in A$ and some $d_\ell=\gamma_\ell m/q_\ell$ with γ_ℓ coprime with m. Then the element $g=\Pi_{\ell=1}^r\bar{g}_{k_\ell}$ is of the form wy^d where $w\in A$ and $d=d_1+\cdots+d_r$ is coprime with m. Thus m divides $|g|$ and $z=g^{|g|/m}$ is an element of order m of the form $w'y^e$ with e coprime with m. From Lemma 3.5 we know that $\langle x_1,\ldots,x_t,h_1,\ldots,h_s\rangle=A$ and conclude that $(\langle x_1,\ldots,x_t,h_1,\ldots,h_s\rangle,\langle z\rangle)\in\mathscr{D}_G^m$.

We now prove the last part of the proposition. Suppose that the algorithm does not err and denote (M,z) its output. Then z has order m and $\langle M\rangle$ is an abelian subgroup of G of order coprime with m, since the tests at steps 8 and 13 succeeded. Moreover $\langle M\rangle$ is normal in G since $G'\leq\langle M\rangle$. We conclude that $\langle M,z\rangle\in\mathscr{S}$ and $(\langle M\rangle,\langle z\rangle)\in\mathscr{D}^m_{\langle M,z\rangle}$. ∎

We now present an algorithm computing a standard decomposition of any group in $\mathscr{S}$.

Theorem 3.7. *There exists a deterministic algorithm that, on an input G in the class $\mathscr{S}$ given as a black box, outputs an element $z\in G$ and a set M of elements in G such*

that $(\langle M\rangle, \langle z\rangle)$ *is a standard decomposition of* G*. The time complexity of this algorithm is* $O(|G|^{1/2+o(1)})$*.*

Proof. The algorithm is as follows. Let G be a group in the class $\mathscr{S}$, input as a black box with generating set $\{g_1, \ldots, g_s\}$ where $s = O(\log |G|)$.

We first compute $|g_i|$ for each $i \in \{1, \ldots, s\}$ using Shanks' algorithm. Let $\bar{m}$ be the least common multiple of the s integers $|g_1|, \ldots, |g_s|$. We compute the set S of divisors of $\bar{m}$, and denote $m_1 < m_2 < \cdots < m_r$ the elements of S in increasing order.

For i from 1 to r we run the procedure FIND-DECOMPOSITION(G, m_i) on the set $\{g_1, \ldots, g_s\}$ and m_i, and obtain an error message or an output $(\langle M_i\rangle, z_i)$. Let n be the maximum value of the quantity $m_i|\langle M_i\rangle|$ over all the i's such that the output is not an error message (we will show that for at least one value of i the output is not an error message so n is well defined). Notice that computing $|M_i|$ can be done using the deterministic algorithm by Buchmann and Schmidt [5] that computes the order of any abelian group K in time $\tilde{O}(|K|^{1/2})$. Finally the algorithm takes the smallest integer $i_0 \in \{1, \ldots, r\}$ such that $m_{i_0}|M_{i_0}| = n$, and then outputs z_{i_0} and M_{i_0}.

We now analyze this algorithm. First of all notice that for any m such that $\mathscr{D}_G^m$ is not empty, this integer m is in S since m divides $\bar{m}$. By Proposition 3.6, if $\mathscr{D}_G^{m_i}$ is not empty then the procedure FIND-DECOMPOSITION(G, m_i) outputs an element $(\langle M_i\rangle, \langle z_i\rangle) \in \mathscr{D}_G^{m_i}$ and then $m_i|\langle M_i\rangle| = |G|$. Conversely, and again by Proposition 3.6, if the procedure FIND-DECOMPOSITION(G, m_i) outputs (M_i, z_i), then $m_i|\langle M\rangle| = |\langle z_i, M_i\rangle| \leq |G|$. Thus n is well defined and is equal to the order of G. Finally, trying all the elements of S gives clearly the minimal m such that $\mathscr{D}_G^m$ is not empty. Then $(\langle M_{i_0}\rangle, z_{i_0})$ is a standard decomposition of G. The time complexity of the algorithm is shown to be $|G|^{1/2+o(1)}$ using Proposition 3.6 and the following two facts. First, computing the set S can be done in $\tilde{O}(|G|^{1/2})$ time. Second, the number of divisors of any integer k has for upper bound $O(k^\varepsilon)$ for any positive constant ε (see for example [8]). Since $\bar{m} \leq |G|$ we conclude that $r = |G|^{o(1)}$. ∎

4. Testing Conjugacy

In this section we study the automorphism group of any abelian group and describe how to decide whether two automorphisms are conjugate.

Let A be a finite abelian group. Then A is the direct product of all its Sylow subgroups. Since $Aut(A)$ is the direct product of the automorphism groups of the Sylow subgroups, we can assume without loss of generality that A is an abelian p-group for some prime p. In this section we suppose that A is isomorphic to the group $\mathbb{Z}_{p^{e_1}} \times \cdots \times \mathbb{Z}_{p^{e_s}}$, for some positive integers s and $e_1 \leq e_2 \leq \ldots \leq e_s$. Let $(g_1, \ldots, g_s)$ be a basis of A, i.e. s elements of A such that the order of each g_i is p^{e_i} and such that $A = \langle g_1\rangle \times \cdots \times \langle g_s\rangle$.

4.1. Automorphisms of an abelian group

We first introduce a matricial characterization of the group $Aut(A)$ and study its structure.

Let ψ be an endomorphism of A and, for each $j \in \{1, \ldots, s\}$, denote $\psi(g_j) = g_1^{u_{1j}} \ldots g_s^{u_{sj}}$ where each u_{ij} is in the set $\{0, \ldots, p^{e_i} - 1\}$. The values u_{ij}, which can be seen as an integer matrix (u_{ij}) of size $s \times s$, fully define the endomorphism ψ. However the converse is not true: an arbitrary integer matrix (u_{ij}) of size $s \times s$ with each value u_{ij} in $\{0, \ldots, p^{e_i} - 1\}$ does

not necessarily define an endomorphism of A, because ψ should be a homomorphism, and not only a linear map. It is easy to give necessary and sufficient conditions for these values u_{ij} to define an endomorphism of A: $p^{e_i - e_{min(i,j)}}$ should divide u_{ij} for any $i, j \in \{1, \ldots, s\}$.

More precisely, define $M(A)$ as the following set of integer matrices.

$$M(A) = \left\{ (u_{ij}) \in \mathbb{Z}^{s \times s} \mid 0 \leq u_{ij} < p^{e_i} \text{ and } p^{e_i - e_{min(i,j)}} \text{ divides } u_{ij} \text{ for all } i, j \in \{1, \ldots, s\} \right\}.$$

Given U and U' in $M(A)$ we also define the multiplication $*$ as follows: $U * U'$ is the integer matrix W of size $s \times s$ such that $w_{ij} = (\sum_{k=1}^{s} u_{ik} u'_{kj} \bmod p^{e_i})$ for $i, j \in \{1, \ldots, s\}$, i.e. after computing the usual matrix multiplication UU', each entry is reduced modulo p^{e_i}, where i is the row of the entry. Let $R(A)$ be the set $R(A) = \{U \in M(A) \mid det(U) \not\equiv 0 \bmod p\}$. Ranum has shown [17] that the set $R(A)$ with the product operation $*$ is a group isomorphic to the group of automorphisms of A. Notice that a canonical isomorphism between $R(A)$ and $Aut(A)$ follows from the choice of a basis for A. An important example is the case $A = \mathbb{Z}_p^s$ for some integer s, for which $M(A)$ is the set of matrices of size $s \times s$ over the finite field $\mathbb{Z}_p$ and $R(A)$ is the general linear group $GL_s(p)$ of invertible matrices of size $s \times s$ over $\mathbb{Z}_p$.

Let us write $A \cong H_1 \times \cdots \times H_t$ with $H_i = \mathbb{Z}_{p^{f_i}}^{k_i}$ where $f_1 < f_2 < \cdots < f_t$ are positive strictly increasing integers and $k_1, \ldots, k_t$ are positive integers. Any matrix $M \in R(A)$ has t diagonal blocks $D_1, \ldots, D_t$ with $D_i \in GL_{k_i}(p)$ for $i \in \{1, \ldots, t\}$. Let Ψ be the map from $R(A)$ to $GL_s(p)$ such that any matrix $M \in R(A)$ is mapped as follows: the entries in the diagonal blocks are reduced modulo p; the other entries are set to zero. It is easy to show that Ψ is a group homomorphism from $Aut(A)$ to $GL_s(r)$. Let $N(A)$ denote its kernel and $V(A)$ denote its image. It is easy to see that N(A) is a subgroup of $R(A)$ of order p^r for some positive integer r, and that $V(A)$ is the subgroup of $GL_s(p)$ consisting of all the block diagonal matrices of the form $diag(D_1, \ldots, D_t)$ with $D_i \in GL_{k_i}(p)$ for $i \in \{1, \ldots, t\}$. We refer to [17] and to the full version of our paper for further details.

4.2. Testing conjugacy in $R(A)$

In this subsection we consider the following computational problem and present an efficient algorithm solving it.

CONJUGACY
INPUT: an abelian p-group A and two matrices U_1 and U_2 in $R(A)$ such that
$$\textit{the orders of } U_1 \textit{ and } U_2 \textit{ are coprime with } p \tag{4.1}$$
OUTPUT: an element $U \in R(A)$ such that $U * U_1 = U_2 * U$ if such an element exists

Trying all the possibilities for U requires $|R(A)|$ trials. Since for example in the case $A = \mathbb{Z}_{p^k}^s$ with p and k constant the bound $|R(A)| = \Theta(|A|^{\log |A|})$ holds, such a naive approach is not efficient. However, notice that in the case $A = \mathbb{Z}_p^s$ the group A has more than the structure of an abelian group: A is a vector space over the field $\mathbb{Z}_p$ and then $R(A) = GL_s(p)$ as explained above. A mathematical criterion for the conjugacy of matrices in $GL_s(p)$ (even without the condition (4.1) on their orders) is known: two matrices are conjugate if and only if their canonical rational forms are equal. Since the canonical rational form of a matrix can be computed efficiently [21], this gives an algorithm solving the problem CONJUGACY in time polynomial in $\log |A|$. However, when A has no vector space structure, there is no known simple mathematical criterion for the conjugacy of matrices and, to our knowledge, no algorithm faster than the above naive approach is known, even for the case

where $A = \mathbb{Z}_{p^2}^s$. We now show that with the additional condition (4.1) on the order of U_1 and U_2 there exists an algorithm solving the problem CONJUGACY in time polynomial in $\log|A|$ for any abelian p-group A.

Our algorithm is based on the following proposition, which is a generalization of an argument by Pomfret [16].

Proposition 4.1. *Let A be an abelian p-group and U_1, U_2 be two matrices in $R(A)$ of order coprime with p. Then U_1 and U_2 are conjugate in $R(A)$ if and only if $\Psi(U_1)$ and $\Psi(U_2)$ are conjugate in $V(A)$. Moreover if U_1 and U_2 are conjugate in $R(A)$ then for any $X \in R(A)$ such that $\Psi(U_1) = \Psi(X)^{-1}\Psi(U_2)\Psi(X)$ there exists a matrix $Y \in N(A)$ such that $X * Y * U_1 = U_2 * X * Y$.*

Proof. For brevity we omit the symbol $*$ when denoting multiplications in $R(A)$. Since Ψ is an homomorphism, if U_1 and U_2 are conjugate in $R(A)$ then $\Psi(U_1)$ and $\Psi(U_2)$ are conjugate in $V(A)$. Now suppose that $\Psi(U_1)$ and $\Psi(U_2)$ are conjugate in $V(A)$. Since the image of Ψ is $V(A)$, there exists some $X \in R(A)$ such that $\Psi(U_1) = \Psi(X)^{-1}\Psi(U_2)\Psi(X)$ and thus $U_1 = X^{-1}U_2XM$ for some $M \in N(A)$. Then $\langle U_1\rangle N(A) = \langle X^{-1}U_2X\rangle N(A)$ (since $N(A)$ is a normal subgroup of $R(A)$) and the two subgroups $\langle U_1\rangle$ and $\langle X^{-1}U_2X\rangle$ are Hall subgroups of the group $\langle U_1\rangle N(A)$. Moreover since $\langle U_1\rangle N(A)$ is a cyclic extension of the p-group $N(A)$, this is a solvable group. Then, from Theorem 2.1, this implies that the two subgroups $\langle U_1\rangle$ and $\langle X^{-1}U_2X\rangle$ are conjugate in $\langle U_1\rangle N(A)$ and thus there exists an element $Y \in \langle U_1\rangle N(A)$ and some $r > 0$ such that $Y^{-1}X^{-1}U_2XY = U_1^r$. Without loss of generality Y can be taken in $N(A)$. Thus $\Psi(U_1) = \Psi(X)^{-1}\Psi(U_2)\Psi(X) = \Psi(U_1)^r$. Since the order of the kernel of Ψ is coprime with the order of U_1, the matrices U_1 and $\Psi(U_1)$ have the same order, and thus $U_1 = U_1^r$. We conclude that $Y^{-1}X^{-1}U_2XY = U_1$. The matrices U_1 and U_2 are thus conjugate in $R(A)$. The second part of the theorem follows from the observation that X can be chosen in an arbitrary way. ∎

We now present our algorithm.

Theorem 4.2. *There exists a deterministic algorithm that solves the problem* CONJUGACY *in time polynomial in* $\log|A|$.

Proof. The algorithm is as follows.

Given U_1 and U_2 in $R(A)$ satisfying Condition (4.1), we first compute the two matrices $V_1 = \Psi(U_1)$ and $V_2 = \Psi(U_2)$ in $V(A)$. Then we check the conjugacy of V_1 and V_2 in $V(A)$ using the following approach. Let $D_i(V_1)$ (resp. $D_i(V_2)$) be the i-th diagonal block of V_1 (resp. V_2) for $i \in \{1, \ldots, t\}$, i.e. a matrix in $GL_{k_i}(p)$. The matrices V_1 and V_2 are conjugate in $V(A)$ if and only if the blocks $D_i(V_1)$ and $D_i(V_2)$ are conjugate in $GL_{k_i}(p)$ for each $i \in \{1, \ldots, t\}$, that is, if $D_i(V_1)$ and $D_i(V_2)$ have the same rational normal form. The rational normal form of matrices of size $n \times n$ (and transformation matrices) over any finite field can be computed using $O(n^4)$ field operations (see for example [21]). Thus we can decide in time polynomial in $\log|A|$ whether $D_i(V_1)$ and $D_i(V_2)$ are conjugate for all $i \in \{1, \ldots, t\}$. If this is not the case then we conclude that U_1 and U_2 are not conjugate in $R(A)$ from Proposition 4.1. Otherwise U_1 and U_2 are conjugate in $R(A)$ and the remaining of the proof shows how to compute a matrix $U \in R(A)$ such that $U * U_1 = U_2 * U$.

We compute transformation matrices $T_i \in GL_{k_i}(p)$, for $i \in \{1, \ldots, t\}$, such that $T_iD_i(V_1) = D_i(V_2)T_i$ using, for example, again the algorithm [21]. Then we take any matrix X in $R(A)$ such that $\Psi(X) = diag(T_1, \ldots, T_t)$, e.g. the matrix X in $R(A)$ with diagonal blocks equal to $T_1, \ldots, T_t$ and zero everywhere else. We finally determine a solution Y in

$N(A)$ of the matrix equation $X*Y*U_1 = U_2*X*Y$. Such solution exists by Proposition 4.1. To do this, we write the general form of an element Y of $N(A)$ using s^2 variables y_{ij}: the entry corresponding to the i-th row and the j-th column of Y, for $i, j \in \{1, \ldots, s\}$, is of the form $(1 + py_{ij})$ if $i = j$ and is of the form $p^{d_{ij}} y_{ij}$ for some appropriate nonnegative integer d_{ij} otherwise. Then the equation $X * Y * U_1 = U_2 * X * Y$ can be rewritten as the following system of s^2 linear modular equations of s^2 variables y_{ij}: $\sum_{i,j=1}^{s} \alpha_{ij}^{(k,\ell)} y_{ij} \equiv \beta^{(k,\ell)} \bmod p^{e_k}$ for $1 \leq k, \ell \leq s$, where $\alpha_{ij}^{(k,\ell)}$ and $\beta^{(k,\ell)}$ are known. Now we add on each modular equation a new variable $z_{k\ell}$ with coefficient p^{e_k}. This transforms the above system into the following system of s^2 linear Diophantine solutions of $2s^2$ variables: $\sum_{i,j=1}^{s} \alpha_{ij}^{(k,\ell)} y_{ij} + p^{e_k} z_{k\ell} = \beta^{(k,\ell)}$ for $1 \leq k, \ell \leq s$. It is known that any system of linear Diophantine equations with n_1 equations and n_2 variables can be solved in time polynomial in n_1, n_2 and $\log N$, where N is the largest coefficient appearing in the system [6]. Then a solution $Y \in N(A)$ of the equation $X * Y * U_1 = U_2 * X * Y$ can be computed in time polynomial in $\log |A|$. The output of the algorithm is the matrix $X * Y$. ∎

5. Our Algorithm

In this section we give a proof of Theorem 1.2. We first present the following rather simple result that shows necessary and sufficient conditions for the isomorphism of two groups in $\mathscr{S}$.

Proposition 5.1. *Let G and H be two groups in $\mathscr{S}$. Let $(A_1, \langle y_1 \rangle)$ and $(A_2, \langle y_2 \rangle)$ be standard decompositions of G and H respectively and let φ_1 (resp. φ_2) be the action by conjugation of y_1 on A_1 (resp. of y_2 on A_2). The groups G and H are isomorphic if and only if the following three conditions hold: (i) $A_1 \cong A_2$; and (ii) $|y_1| = |y_2|$; and (iii) there exists an integer $k \in \{1, \ldots, |y_1|\}$ coprime with $|y_1|$ and an isomorphism $\psi : A_1 \to A_2$ such that $\varphi_1 = \psi^{-1} \varphi_2^k \psi$.*

Proof. First notice that for a group G in $\mathscr{S}$, the integer $\gamma(G)$ is a group invariant. Now suppose that G and H are two isomorphic groups in $\mathscr{S}$ with standard decomposition respectively $(A_1, \langle y_1 \rangle)$ and $(A_2, \langle y_2 \rangle)$. Then $|y_1| = |y_2| = \gamma(G) = \gamma(H)$. Denote by ψ an isomorphism from G to H and notice that $(\psi(A_1), \psi(y_1)) \in \mathscr{D}_H^{\gamma(H)}$. From Lemma 3.4 this implies that $\psi(A_1) = A_2$ and, in particular, $A_1 \cong A_2$. The element $\psi(y_1)$ can be written as $z y_2^k$ for some $z \in A_2$ and some integer $k \in \{1, \ldots, \gamma(H)\}$ coprime with $\gamma(H)$. By definition of φ_1, for any $x \in A_1$ the relation $y_1 x = (y_1 x y_1^{-1}) y_1 = \varphi_1(x) y_1$ holds. Applying ψ to each term gives $z y_2^k \psi(x) = \psi(\varphi_1(x)) z y_2^k$ and then $\varphi_2^k(\psi(x)) z y_2^k = \psi(\varphi_1(x)) z y_2^k$ for any $x \in A_1$. Thus $\varphi_2^k = \psi \varphi_1 \psi^{-1}$.

Now consider two groups G and H in $\mathscr{S}$ satisfying the conditions (i), (ii) and (iii) of the statement of the theorem. Denote $m = |y_1| = |y_2|$. Let μ be the map from G to H such that $\mu(x y_1^j) = \psi(x_1) y_2^{kj}$ for any x in A_1 and any $j \in \{0, \ldots, m-1\}$. The map μ is clearly a bijection from G to H. We now show that μ is a homomorphism, and thus an isomorphism from G to H. Let x and x' be two elements of A_1 and let j and j' be two elements of $\in \{0, \ldots, m-1\}$. Then $\mu(x y_1^j x' y_1^{j'}) = \mu(x \varphi_1^j(x') y_1^{j+j'}) = \psi(x \varphi_1^j(x')) y_2^{k(j+j')} = \psi(x) \psi(\varphi_1^j(x')) y_2^{k(j+j')}$. Now the relation $\mu(x y_1^j) \mu(x' y_1^{j'}) = \psi(x) y_2^{kj} \psi(x') y_2^{kj'} = \psi(x) \varphi_2^{kj}(\psi(x')) y_2^{k(j+j')}$ holds. Condition (iii) of the statement of the theorem implies that $\psi(\varphi_1^j(x')) = \varphi_2^{kj}(\psi(x'))$ and thus $\mu(x y_1^j x' y_1^{j'}) = \mu(x y_1^j) \mu(x' y_1^{j'})$. ∎

We now present our proof of Theorem 1.2.

Proof of Theorem 1.2. Suppose that G and H are two groups in the class $\mathscr{S}$. Denote $n = min(|G|, |H|)$ and $\gamma = min(\gamma(G), \gamma(H))$. Without loss of generality let us suppose that $|G| = |H|$. In order to test whether these two groups are isomorphic, we first run the algorithm of Theorem 3.7 on the inputs G and H and obtain outputs (S_1, y_1) and (S_2, y_2) such that $(\langle S_1\rangle, \langle y_1\rangle)$ and $(\langle S_2\rangle, \langle y_2\rangle)$ are standard decompositions of G and H respectively. The running time of this algorithm is $O(n^{1/2+o(1)})$ by Theorem 3.7. Denote $A_1 = \langle S_1\rangle$ and $A_2 = \langle S_2\rangle$.

We then check whether $|y_1| = |y_2|$. If $|y_1| \neq |y_2|$ we conclude that G and H are not isomorphic by Proposition 5.1. Otherwise notice that $|y_1| = |y_2| = \gamma$. Then we compute a basis $(g_1, \ldots, g_s)$ of A_1 and a basis $(h_1, \ldots, h_t)$ of A_2 using the algorithm by Buchmann and Schmidt [5]. The running time of this step is $\tilde{O}(n^{1/2})$. Given these bases it is easy to check the isomorphism of A_1 and A_2: the groups A_1 and A_2 are isomorphic if and only if $s = t$ and $|g_i| = |h_i|$ for each $i \in \{1, \ldots, s\}$. If $A_1 \ncong A_2$ we conclude that G and H are not isomorphic by Proposition 5.1.

Now suppose that $A_1 \cong A_2$ (and then $s = t$) and denote $R = R(A_1) = R(A_2)$. We want to decide whether the action by conjugation φ_1 of y_1 on A_1 and the action by conjugation φ_2 of y_2 on A_2 satisfy Condition (iii) in Proposition 5.1. Let $p_1^{d_1} \cdots p_r^{d_r}$ be the prime power decomposition of $|A_1| = |A_2|$, with $p_1 < \cdots < p_d$ and denote P_i the Sylow p_i-subgroup of A_1 for each $i \in \{1, \ldots, r\}$. We compute the matrix M_1 in R corresponding to the automorphism φ_1 of A_1 with respect to the basis $(g_1, \ldots, g_s)$. More precisely let us denote $\varphi_1(g_i) = y_1 g_i y_1^{-1} = g_1^{u_{i1}} \cdots g_j^{u_{is}}$ for each $i \in \{1, \ldots, s\}$. The values u_{ij} for each i can be found by using the algorithm of Proposition 3.3 on the input $y_1 g_i y_1^{-1}$. Then the matrix $M_1 = (u_{ij})$ can be computed in time $\tilde{O}(n^{1/2})$. Similarly we compute the matrix $M_2 \in R$ corresponding to the automorphism φ_2 of A_2 with respect to the basis $(h_1, \ldots, h_s)$. A key observation is that M_1 and M_2 are block diagonal, consisting in r blocks. More precisely the i-th block is a matrix in $R(P_i)$.

Finally for each integer $k \in \{1, \ldots, \gamma\}$ coprime with γ, we test whether M_1 and M_2^k are conjugate in R. This is done by using the algorithm of Theorem 4.2 to check whether, for each $i \in \{1, \ldots, r\}$, the i-th block of M_1 is conjugate to the i-th block of M_2 in $R(P_i)$. If there is no k such that M_1 and M_2^k are conjugate in R we conclude that G and H are not isomorphic. Otherwise we take one value k such that M_1 and M_2^k are conjugate and compute an explicit block diagonal matrix X in R such that $M_1 = X^{-1} M_2^k X$. This can be done in time polynomial in $\log n$ by Theorem 4.2. The matrix X is naturally associated to an isomorphism ψ from A_1 to A_2 through the bases $(g_1, \ldots, g_s)$ and $(h_1, \ldots, h_s)$. The map $\mu : G \to H$ defined as $\mu(x y_1^j) = \psi(x) y_2^{kj}$ for any $x \in A_1$ and any $j \in \{0, \ldots, \gamma - 1\}$ is then an isomorphism from G to H (see the proof of Proposition 5.1 for details). The total complexity of this final step is $O(\gamma \log^c n)$ for some constant c.

The time complexity of this algorithm is $O(\gamma \log^c n) + O(n^{1/2+o(1)}) \leq (\sqrt{n} + \gamma)^{1+o(1)}$. ∎

Acknowledgments

The author is grateful to Yoshifumi Inui for many discussions on similar topics. He also thanks Igor Shparlinski and Erich Kaltofen for helpful comments.

References

[1] V. Arvind and J. Torán. Solvable group isomorphism. In *Proceedings of the 19th IEEE Conference on Computational Complexity*, pages 91–103, 2004.

[2] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th annual ACM symposium on Theory of computing*, pages 421–429, 1985.

[3] L. Babai. Local expansion of vertex-transitive graphs and random generation in finite groups. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 164–174, 1991.

[4] L. Babai and E. Szemerédi. On the complexity of matrix group problems I. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, pages 229–240, 1984.

[5] J. Buchmann and A. Schmidt. Computing the structure of a finite abelian group. *Mathematics of Computation*, 74(252):2017–2026, 2005.

[6] T. J. Chou and G. E. Collins. Algorithms for the solution of systems of linear diophantine equations. *SIAM Journal on Computing*, 11(4):687–708, 1982.

[7] M. H. Garzon and Y. Zalcstein. On isomorphism testing of a class of 2-nilpotent groups. *Journal of Computer and System Sciences*, 42(2):237–248, 1991.

[8] G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers*. Oxford Science Publications, 1979.

[9] D. F. Holt, B. Eick, and E. A. O'Brien. *Handbook of computational group theory*. Chapman & Hall / CRC, 2005.

[10] R. Kannan and A. Bachem. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM Journal on Computing*, 8(4):499–507, 1979.

[11] T. Kavitha. Linear time algorithms for abelian group isomorphism and related problems. *Journal of Computer and System Sciences*, 73(6):986–996, 2007.

[12] J. Köbler, J. Torán, and U. Schöning. *The graph isomorphism problem: its structural complexity*. Birkhäuser, 1993.

[13] R. J. Lipton, L. Snyder, and Y. Zalcstein. The complexity of word and isomorphism problems for finite groups. Technical report, John Hopkins, 1976.

[14] G. Miller. On the $n^{\log n}$ isomorphism technique. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 51–58, 1978.

[15] E. A. O'Brien. Isomorphism testing for p-groups. *Journal of Symbolic Computation*, 17:133–147, 1994.

[16] J. Pomfret. Similarity of matrices over finite rings. *Proceedings of the American Mathematical Society*, 37(2):421–422, 1973.

[17] A. Ranum. The group of classes of congruent matrices with application to the group of isomorphisms. *Transactions of the American Mathematical Society*, 8(1):71–91, 1907.

[18] S. Shanks. Class number, a theory of factorization and generata. *Proceedings of Symposia in Pure Mathematics*, 20(419-440), 1969.

[19] V. Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2005.

[20] M. Smith. *Computing automorphisms of finite soluble groups*. PhD thesis, Australian National University, 1994.

[21] A. K. Steel. Algorithm for the computation of canonical forms of matrices over fields. *Journal of Symbolic Computation*, 24(3/4):409–432, 1997.

[22] N. Vikas. An $O(n)$ algorithm for abelian p-group isomorphism and an $O(n \log n)$ algorithm for abelian group isomorphism. *Journal of Computer and System Sciences*, 53(1):1–9, 1996.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 637–648
www.stacs-conf.org

ON APPROXIMATING MULTI-CRITERIA TSP

BODO MANTHEY [1]

[1] Saarland University, Computer Science, Postfach 151150, 66041 Saarbrücken, Germany
E-mail address: manthey@cs.uni-sb.de

Abstract. We present approximation algorithms for almost all variants of the multi-criteria traveling salesman problem (TSP), whose performances are independent of the number k of criteria and come close to the approximation ratios obtained for TSP with a single objective function.

We present randomized approximation algorithms for multi-criteria maximum traveling salesman problems (Max-TSP). For multi-criteria Max-STSP, where the edge weights have to be symmetric, we devise an algorithm that achieves an approximation ratio of $2/3 - \varepsilon$. For multi-criteria Max-ATSP, where the edge weights may be asymmetric, we present an algorithm with an approximation ratio of $1/2 - \varepsilon$. Our algorithms work for any fixed number k of objectives. To get these ratios, we introduce a decomposition technique for cycle covers. These decompositions are optimal in the sense that no decomposition can always yield more than a fraction of $2/3$ and $1/2$, respectively, of the weight of a cycle cover. Furthermore, we present a deterministic algorithm for bi-criteria Max-STSP that achieves an approximation ratio of $61/243 \approx 1/4$.

Finally, we present a randomized approximation algorithm for the asymmetric multi-criteria minimum TSP with triangle inequality (Min-ATSP). This algorithm achieves a ratio of $\log n + \varepsilon$. For this variant of multi-criteria TSP, this is the first approximation algorithm we are aware of. If the distances fulfil the γ-triangle inequality, its ratio is $1/(1-\gamma) + \varepsilon$.

1. Multi-Criteria Traveling Salesman Problem

Traveling Salesman Problem. The traveling salesman problem (TSP) is one of the most famous combinatorial optimization problems. Given a graph, the goal is to find a Hamiltonian cycle (also called a *tour*) of maximum or minimum weight (Max-TSP or Min-TSP). An instance of Max-TSP is a complete graph $G = (V, E)$ with edge weights $w : E \to \mathbb{Q}_+$. The goal is to find a Hamiltonian cycle of maximum weight. The weight of a Hamiltonian cycle (or, more general, of any set of edges) is the sum of the weights of its edges. If G is undirected, we have Max-STSP (symmetric TSP). If G is directed, we obtain Max-ATSP (asymmetric TSP). An instance of Min-TSP is also a complete graph G with edge weights w that fulfil the triangle inequality: $w(u, v) \leq w(u, x) + w(x, v)$ for all $u, v, x \in V$. The goal is to find a tour of minimum weight. We have Min-STSP if G is undirected and Min-ATSP if G is directed. In this paper, we only consider the latter. If we restrict the instances to fulfil the γ-triangle inequality ($w(u, v) \leq \gamma \cdot (w(u, x) + w(x, v))$ for all distinct $u, v, x \in V$

Key words and phrases: Approximation algorithms, traveling salesman, multi-criteria optimization.

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 637-648
http://drops.dagstuhl.de/opus/volltexte/2009/1853

and $\gamma \in [\frac{1}{2}, 1)$), then we obtain Min-$\gamma$-ATSP. All variants introduced are NP-hard and APX-hard (Min-γ-ATSP is hard for $\gamma > \frac{1}{2}$). Thus, we have to content ourselves with approximate solutions. The currently best approximation algorithms for Max-STSP and Max-ATSP achieve approximation ratios of 61/81 [7] and 2/3 [14]. Min-ATSP can be approximated with a factor of $\frac{2}{3} \cdot \log_2 n$, where n is the number of vertices of the instance [11]. Min-γ-ATSP allows for an approximation ratio of $\min\{\frac{\gamma}{1-\gamma}, \frac{1+\gamma}{2-\gamma-\gamma^3}\}$ [5, 6].

Cycle covers are often used for designing approximation algorithms for the TSP [5, 14, 11, 6, 15, 7]. A cycle cover is a set of vertex-disjoint cycles such that every vertex is part of exactly one cycle. The idea is to compute an initial cycle cover and then to join the cycles to obtain a Hamiltonian cycle. This is called *subtour patching* [13]. Hamiltonian cycles are special cases of cycle covers that consist of a single cycle. Thus, the weight of a maximum-weight cycle cover bounds the weight of a maximum-weight Hamiltonian cycle from above, and the weight of a minimum-weight cycle cover is a lower bound for the weight of a minimum-weight Hamiltonian cycle. In contrast to Hamiltonian cycles, cycle covers of optimal weight can be computed efficiently by reduction to matching problems [1].

Multi-Criteria Optimization. In many optimization problems, there is more than one objective function. This is also the case for the TSP: We might want to minimize travel time, expenses, number of flight changes, etc., while maximizing, e.g., the number of sights along the way. This leads to k-criteria variants of the TSP (k-C-Max-STSP, k-C-Max-ATSP, k-C-Min-STSP, k-C-Min-ATSP for short; if the number of criteria does not matter, we will also speak of MC-Max-STSP etc.). With respect to a single criterion, the term "optimal solution" is well-defined. However, if several criteria are involved, there is no natural notion of a best choice, and we have to be content with trade-off solutions. The goal of multi-criteria optimization is to cope with this dilemma. To transfer the concept of optimal solutions to multi-criteria problems, the notion of *Pareto curves* was introduced (cf. Ehrgott [9]). A Pareto curve is a set of solutions that can be considered optimal.

We introduce the following terms only for maximization problems. After that, we briefly state the differences for minimization problems. An instance of k-C-Max-TSP is a complete graph G with edge weights $w_1, \ldots, w_k : E \to \mathbb{Q}_+$. A tour H *dominates* another tour H' if $w_i(H) \geq w_i(H')$ for all $i \in [k] = \{1, \ldots, k\}$ and $w_i(H) > w_i(H')$ for at least one i. This means that H is strictly preferable to H'. A *Pareto curve* of solutions contains all solutions that are not dominated by another solution. For other maximization problems, k-criteria variants are defined analogously.

Unfortunately, Pareto curves cannot be computed efficiently in many cases: First, they are often of exponential size. Second, they are often NP-hard to compute even for otherwise easy optimization problems. Third, the TSP is NP-hard already with one objective function, and optimization problems do not become easier with more objectives involved. Therefore, we have to be satisfied with approximate Pareto curves.

For simpler notation, let $w(H) = (w_1(H), \ldots, w_k(H))$. Inequalities are meant component-wise. A set $\mathcal{P}$ of Hamiltonian cycles of V is called an α *approximate Pareto curve* for (G, w) if the following holds: For every tour H', there exists a tour $H \in \mathcal{P}$ with $w(H) \geq \alpha w(H')$. We have $\alpha \leq 1$, and a 1 approximate Pareto curve is a Pareto curve. (This is not precisely true if there are several solutions whose objective values agree. But this is inconsequential here, and we will not elaborate on it for the sake of clarity.)

An algorithm is called an α approximation algorithm if, given G and w, it computes an α approximate Pareto curve. It is called a randomized α approximation if its success

probability is at least $1/2$. This success probability can be amplified to $1-2^{-m}$ by executing the algorithm m times and taking the union of all sets of solutions. (We can also remove solutions from this union that are dominated by other solutions in the union, but this is not required by the definition of an approximate Pareto curve.) Again, the concepts can be transfered easily to other maximization problems.

Papadimitriou and Yannakakis [18] showed that $(1-\varepsilon)$ approximate Pareto curves of size polynomial in the instance size and $1/\varepsilon$ exist. The technical requirement for the existence is that the objective values of all solutions for an instance X are bounded from above by $2^{p(N)}$ for some polynomial p, where N is the size of X. This is fulfilled in most optimization problems and in particular in our case. However, they only prove the existence, and for many optimization problems it is unclear how to actually find an approximate Pareto curve.

A *fully polynomial time approximation scheme* (FPTAS) for a multi-criteria optimization problem computes $(1-\varepsilon)$ approximate Pareto curves in time polynomial in the size of the instance and $1/\varepsilon$ for all $\varepsilon > 0$. Multi-criteria maximum-weight matching admits a *randomized FPTAS* [18], i.e., the algorithm succeeds in computing a $(1-\varepsilon)$ approximate Pareto curve with constant probability. This randomized FPTAS yields also a randomized FPTAS for the multi-criteria maximum-weight cycle cover problem [17].

To define Pareto curves and approximate Pareto curves also for minimization problems, in particular for MC-Min-STSP and MC-Min-ATSP, we have to replace all "$\geq$" and "$>$" above by "$\leq$" and "$<$". Furthermore, α approximate Pareto curves are now defined for $\alpha \geq 1$, and an FPTAS has to achieve an approximation ratio of $1+\varepsilon$. There also exists a randomized FPTAS for the multi-criteria minimum-weight cycle cover problem.

Related Work. For an overview of the literature about multi-criteria optimization, including multi-criteria TSP, we refer to Ehrgott and Gandibleux [10]. Angel et al. [2, 3] considered Min-STSP restricted to edge weights 1 and 2. They analyzed a local search heuristic and proved that it achieves an approximation ratio of $3/2$ for $k=2$ and of $\frac{2k}{k-1}$ for $k \geq 3$. Ehrgott [8] considered a variant of MC-Min-STSP, where all objectives are encoded into a single objective by using some norm. He proved approximation ratios between $3/2$ and 2 for this problem, where the ratio depends on the norm used.

Manthey and Ram [17] designed a $(2+\varepsilon)$ approximation algorithm for MC-Min-STSP and an approximation algorithm for MC-Min-γ-ATSP, which achieves a constant ratio but works only for $\gamma < 1/\sqrt{3} \approx 0.58$. They left open the existence of approximation algorithms for MC-Max-STSP, MC-Max-ATSP, and MC-Min-ATSP.

Bläser et al. [4] devised the first randomized approximation algorithms for MC-Max-STSP and MC-Max-ATSP. Their algorithms achieve ratios of $\frac{1}{k}+\varepsilon$ for k-C-Max-STSP and $\frac{1}{k+1}+\varepsilon$ for k-C-Max-ATSP. They argue that with their approach, only approximation ratios of $\frac{1}{k \pm O(1)}$ can be achieved, but they conjectured that ratios of $\Omega(1/\log k)$ are possible.

New Results. We devise approximation algorithms for MC-Max-STSP, MC-Max-ATSP, and MC-Min-ATSP. The approximation ratios achieved by our algorithms are independent of the number k of criteria, and they come close to the best approximation ratios known for Max-STSP, Max-ATSP, and Min-ATSP with only a single objective function. Our algorithms work for any number k of criteria.

First, we solve the conjecture of Bläser et al. [4] affirmatively. We even prove a stronger result: For MC-Max-STSP, we achieve a ratio of $2/3-\varepsilon$, while for MC-Max-ATSP, we achieve a ratio of $1/2-\varepsilon$ (Section 4). Already for $k=2$, this is an improvement from

$\frac{1}{2} - \varepsilon$ to $\frac{2}{3} - \varepsilon$ for 2-C-Max-STSP and from $\frac{1}{3} - \varepsilon$ to $\frac{1}{2} - \varepsilon$ for 2-C-Max-ATSP. The general idea of these algorithms is sketched in Section 2. After that, we introduce a decomposition technique in Section 3 that will lead to our algorithms. The running-time of our algorithms is polynomial in the input size for any fixed $\varepsilon > 0$ and any fixed number k of criteria.

Furthermore, we devise a *deterministic* approximation algorithm for 2-C-Max-STSP that achieves a ratio of $61/243 > 1/4$. As a side effect, this proves that for 2-C-Max-STSP, there always exists a single tour that already is a 1/3 approximate Pareto curve.

Finally, we devise the first approximation algorithm for MC-Min-ATSP (Section 6). In addition, our algorithm improves on the algorithm for MC-Min-γ-ATSP by Manthey and Ram [17] for $\gamma > 0.55$, and it is the first approximation algorithm for MC-Min-γ-ATSP for $\gamma \in [1/\sqrt{3}, 1)$. The approximation ratio of our algorithm is $\log n + \varepsilon$ for MC-Min-ATSP, where n is the number of vertices. Furthermore, it is a $\frac{1}{1-\gamma} + \varepsilon$ approximation for MC-Min-γ-ATSP for $\gamma \in [\frac{1}{2}, 1)$. Our algorithm is randomized.

Due to lack of space, most proofs are omitted. For complete proofs, we refer to the full version of this paper [16].

2. Outline and Idea for MC-Max-TSP

For Max-ATSP, we can easily get a 1/2 approximation: We compute a maximum-weight cycle cover, and remove the lightest edge of each cycle. In this way, we obtain a collection of paths. Then we add edges to connect the paths, which yields a Hamiltonian cycle. For Max-STSP, this approach yields a ratio of 2/3.

Unfortunately, this does not generalize to multi-criteria Max-TSP, even though $(1 - \varepsilon)$ approximate Pareto curves of cycle covers can be computed in polynomial time. The reason is that the term "lightest edge" is usually not well defined: An edge that has little weight with respect to one objective might have a huge weight with respect to another objective. Based on this observation, the basic idea behind our algorithms is the following case distinction: First, if every edge of a cycle cover is a *light-weight edge*, i.e., it contributes only little to the overall weight, then removing one edge does not decrease the total weight by too much. We can choose the edges for removal carefully to get an approximate tour.

Second, if there is one edge that is very heavy with respect to one objective (a *heavy-weight edge*), then we contract this edge. We repeat this process until either we have obtained a cycle cover that contains only light-weight edges or we have enough weight for one objective. In the former case, we can use decomposition. In the latter case, we proceed recursively on the remaining graph with $k - 1$ objectives.

In Section 3, we deal with the first case. This includes the definition of when we call an edge a light-weight edge. In Section 4, we present our algorithms, which includes the recursion in case of a heavy-weight edge. The approximation ratios that we achieve come close, i.e., up to an arbitrarily small additive $\varepsilon > 0$, to the 1/2 and 2/3 mentioned above for mono-criterion Max-ATSP and Max-STSP.

3. Decompositions

From any collection P of paths, we obtain a Hamiltonian cycle just by connecting the endpoints of the paths appropriately. Assume that we are given a cycle cover C. If we can find a collection of paths $P \subseteq C$ (by removing one edge of every cycle of C) with

$w(P) \geq \alpha \cdot w(C)$ for some $\alpha \in (0,1]$, then this would yield an approximate solution for Max-TSP. We call these paths P an *α-decomposition of C* for some $\alpha \in (0,1]$ if $w(P) \geq \alpha w(C)$. Not every cycle cover possesses an α-decomposition for every α. Let $k \geq 1$ be the number of criteria. Bläser et al. defined $\alpha_k^d \in [0,1]$ to be the maximum number such that the following holds: every directed cycle cover C with edge weights $w = (w_1, \ldots, w_k)$ that satisfies $w(e) \leq \alpha_k^d \cdot w(C)$ for all $e \in C$ possesses an α_k^d-decomposition. The value $\alpha_k^u \in [0,1]$ is analogously defined for undirected cycle covers. We have $\alpha_1^d = \frac{1}{2}$ and $\alpha_1^u = \frac{2}{3}$. We also have $\alpha_k^u \geq \alpha_k^d$ and $\alpha_k^u \leq \alpha_{k-1}^u$ as well as $\alpha_k^d \leq \alpha_{k-1}^d$.

Bläser et al. [4] proved $\alpha_k^d \geq \frac{1}{k+1}$ and $\alpha_k^u \geq \frac{1}{k}$. Furthermore, they proved the existence of $\Omega(1/\log k)$-decompositions, i.e., $\alpha_k^d, \alpha_k^u \in \Omega(1/\log k)$, which led to their conjecture that $\Omega(1/\log k)$ approximation algorithms might exist. However, their approximation algorithms do not make use of the $\Omega(1/\log k)$ decompositions, and they only achieve ratios of $\frac{1}{k} - \varepsilon$ for k-C-Max-STSP and $\frac{1}{k+1} - \varepsilon$ for k-C-Max-ATSP. In fact, they indicate that approximation ratios of $\frac{1}{k+O(1)}$ are the best that can be proved using their approach. For completeness, we make their decomposition result more precise with the next theorem. In particular, we show that $\alpha_k^d, \alpha_k^u \in \Theta(1/\log k)$, which proves that better approximations require a different decomposition technique. The new decompositions will be introduced later on in this section.

Theorem 3.1. *For all $1 \leq k \in \mathbb{N}$, we have*

$$\frac{1}{0.78 \cdot \log_2 k + \frac{3}{2}} \approx \frac{1}{\frac{9}{8} \cdot \ln k + \frac{3}{2}} \leq \alpha_k^u \leq \frac{1}{\lfloor \log_3 k \rfloor + 1} \approx \frac{1}{0.63 \cdot \log_2 k + 1} \text{ and}$$
$$\frac{1}{1.39 \cdot \log_2 k + 4} \approx \frac{1}{2 \cdot \ln k + 4} \leq \alpha_k^d \leq \frac{1}{\lfloor \log_2 k \rfloor + 2}.$$

In order to obtain constant approximation ratios, independent of k, we have to generalize the concept of decompositions. Let C be a cycle cover, and let $w = (w_1, \ldots, w_k)$ be edge weights. We say that the pair (C, w) is γ-light for some $\gamma \geq 1$ if $w(e) \leq w(C)/\gamma$ for all $e \in C$. In the following, let $\eta_{k,\varepsilon} = \frac{\varepsilon^2}{2 \ln k}$.

Theorem 3.2. *Let ε be arbitrary with $0 < \varepsilon < 1/2$, and let $k \geq 2$ be arbitrary. Let C be a cycle cover, and let $w = (w_1, \ldots, w_k)$ be edge weights such that (C, w) is $1/\eta_{k,\varepsilon}$-light. If C is directed, then there exists a collection $P \subseteq C$ of paths with $w(P) \geq (\frac{1}{2} - \varepsilon) \cdot w(C)$. If C is undirected, then there exists a collection $P \subseteq C$ of paths with $w(P) \geq (\frac{2}{3} - \varepsilon) \cdot w(C)$.*

We know that decompositions exist due to Theorem 3.2. But, in order to use them in approximation algorithms, we have to find them efficiently. In the remainder of this section, we devise devise a simple randomized algorithm for this job. There is also a deterministic algorithm that we call DECOMPOSE with parameters C, w, and ε: C is a cycle cover (directed or undirected), $w = (w_1, \ldots, w_k)$ are k edge weights, and $\varepsilon > 0$. Then DECOMPOSE(C, w, ε) returns a $(\frac{1}{2} - \varepsilon)$- or $(\frac{2}{3} - \varepsilon)$-decomposition $P \subseteq C$, provided that (C, w) is $1/\eta_{k,\varepsilon}$-light. Due to lack of space, we do not describe DECOMPOSE here.

The randomized algorithm exploits Theorem 3.2: Assume that we have a cycle cover C with edge weights w such that (C, w) is $1/\eta_{k,\varepsilon}$-light. We randomly select one edge of every cycle of C for removal and put all remaining edges into P. The probability that P is not a $(\frac{1}{2} - \varepsilon)$- or $(\frac{2}{3} - \varepsilon)$-decomposition (depending on whether C is directed or undirected) is bounded from above by $1/k \leq 1/2$. Thus, we obtain a decomposition with constant probability. We iterate this process until a feasible decomposition has been found. In this way, we get a Las Vegas algorithm with expected linear running-time.

4. Approximation Algorithms for MC-Max-TSP

In this section, MaxCC-Approx denotes the randomized FPTAS for cycle covers. More precisely, let G be a graph (directed or undirected), $w = (w_1, \dots, w_k)$ be edge weights, $\varepsilon > 0$ and $p \in (0, 1]$. Then MaxCC-Approx$(G, w, k, \varepsilon, p)$ yields a $(1-\varepsilon)$-approximate Pareto curve of cycle covers of G with weights w with a success probability of at least $1-p$.

4.1. Multi-Criteria Max-ATSP

Our goal is now either to use decomposition or to reduce the k-criteria instance to a $(k-1)$-criteria instance. To this aim, we put the cart before the horse: Instead of computing Hamiltonian cycles, we assume that they are given. Then we show how to force an algorithm to find approximations to them. To obtain a $1/2-\varepsilon$ approximate Pareto curve, we have to make sure that for every tour $\tilde{H}$, we have a tour H in our set with $w(H) \geq (\frac{1}{2}-\varepsilon) \cdot w(\tilde{H})$. Fix ε with $0 < \varepsilon < \frac{1}{2 \ln k}$, let $\tilde{H}$ be any tour, and let $\beta_i = \max\{w_i(e) \mid e \in \tilde{H}\}$ be the weight of the heaviest edge with respect to the ith objective. Let $\beta = \beta(\tilde{H}) = (\beta_1, \dots, \beta_k)$. We will distinguish two cases.

In the first case, we assume that $\beta \leq (\eta_{k,\varepsilon} - \varepsilon^3) \cdot w(\tilde{H})$, i.e., $\tilde{H}$ does not contain any heavy-weight edges. We modify our edge weights w to w^β as follows:

$$w^\beta(e) = \begin{cases} w(e) & \text{if } w(e) \leq \beta \text{ and} \\ 0 & \text{if } w_i(e) > \beta_i \text{ for some } i. \end{cases}$$

This means that we set all edge weights exceeding β to 0. Since $\tilde{H}$ does not contain any edges whose weight has been set to 0, we have $w(\tilde{H}) = w^\beta(\tilde{H})$. Furthermore, for all subsets C of edges, we have $w^\beta(C) \leq w(C)$. The advantage of w^β is that, if we compute a $(1-\varepsilon)$ approximate Pareto curve $\mathcal{C}^\beta$ of cycle covers with edge weights w^β, we obtain a cycle cover to which we can apply decomposition to obtain a collection P of paths. Then P yields a tour H that approximates $\tilde{H}$. This is stated in the following lemma.

Lemma 4.1. *Let $\varepsilon > 0$ be arbitrary. Let $\tilde{H}$ be a directed tour with $w(e) \leq (\eta_{k,\varepsilon} - \varepsilon^3) \cdot w(\tilde{H})$ for all $e \in \tilde{H}$. Let $\beta = \beta(\tilde{H})$, and let $\mathcal{C}^\beta$ be a $(1-\varepsilon)$ approximate Pareto curve of cycle covers with respect to w^β.*

Then $\mathcal{C}^\beta$ contains a cycle cover C that yields a decomposition $P \subseteq C$ with $w(P) \geq (\frac{1}{2} - 2\varepsilon) \cdot w(\tilde{H})$.

In the second case, we assume that there exists an edge $e = (u, v) \in \tilde{H}$ and an $i \in [k]$ with $w_i(e) > (\eta_{k,\varepsilon} - \varepsilon^3) \cdot w(\tilde{H})$. We put this edge into a set K of edges that we want to have in our cycle cover no matter what. Then we contract the edge e by removing all outgoing edges of u and all incoming edges of v and identifying u and v. In this way, we obtain a slightly smaller tour $\tilde{H}' = \tilde{H} \setminus \{e\}$. Again, there might be an edge $e' \in \tilde{H}'$ and an $i' \in [k]$ with $w_{i'}(e') > (\eta_{k,\varepsilon} - \varepsilon^3) \cdot w_{i'}(\tilde{H}')$. (Since $w(\tilde{H}') \leq w(\tilde{H})$, edges that have not been heavy can now be heavy with respect to $\tilde{H}'$.) We put e' into K, contract e' and recurse. How long can this process go on? There are two cases that can bring it to an end: First, we might get a tour H' that does not have any more heavy-weight edges, i.e., $w(e) \leq (\eta_{k,\varepsilon} - \varepsilon^3) \cdot w(H')$ for all $e \in H'$. In this case, we can apply Lemma 4.1 with decomposition. Second, we might get an $i \in [k]$ with $w_i(K) \geq (\frac{1}{2} - \varepsilon) \cdot w_i(\tilde{H})$, where $\tilde{H}$ is our original tour. Then we have collected enough weight with respect to the ith objective, and we can continue with only $k-1$ objectives. The next lemma bounds the number of edges in K from above.

$\mathcal{P}_{\text{TSP}} \leftarrow$ MaxATSP-Approx$(G, w, k, \varepsilon, p)$
input: directed complete graph $G = (V, E)$, $k \geq 1$, edge weights $w : E \to \mathbb{N}^k$, $\varepsilon > 0$
output: $(\frac{1}{2} - \varepsilon)$ approximate Pareto curve $\mathcal{P}_{\text{TSP}}$ for k-C-Max-ATSP with a success probability of at least $1 - p$
1: **if** $k = 1$ **then**
2: compute a 2/3 approximation $\mathcal{P}_{\text{TSP}}$
3: **else**
4: **for all** subsets $K \subseteq E$ with $|K| \leq f(k, \varepsilon/2)$ such that K is a path cover **do**
5: contract all edges of K to obtain G_K
6: **for all** bounds β of (G_K, w) **do**
7: $\mathcal{C}_{K,\beta} \leftarrow$ MaxCC-Approx$\big(G_K, w^\beta, k, \frac{\varepsilon}{2}, \frac{p}{2n^{2k+2f(k,\varepsilon/2)}}\big)$
8: **for all** $C \in \mathcal{C}_{K,\beta}$ with $w^\beta(e) \leq \eta_{k,\varepsilon/2} \cdot w^\beta(C)$ for all $e \in C$ **do**
9: $P \leftarrow$ Decompose$(C, w^\beta, \varepsilon/2)$
10: add edges to $K \cup P$ to obtain a tour H; add H to $\mathcal{P}_{\text{TSP}}$
11: **for all** $i \leftarrow 1$ **to** k **do**
12: remove the ith objective from w to obtain w'
13: $\mathcal{P}^{K,i}_{\text{TSP}} \leftarrow$ MaxATSP-Approx$(G_K, w', k-1, \frac{\varepsilon}{2}, \frac{p}{2n^{2k+2f(k,\varepsilon/2)}})$
14: **for all** $H' \in \mathcal{P}^{K,i}_{\text{TSP}}$ **do**
15: $H \leftarrow K \cup H'$; add H to $\mathcal{P}_{\text{TSP}}$

Algorithm 1: Approximation algorithm for MC-Max-ATSP.

Lemma 4.2. *After at most* $f(k, \varepsilon) = k \cdot \lceil \frac{\log(1/2+\varepsilon)}{\log(1-\eta_{k,\varepsilon}+\varepsilon^3)} \rceil$ *iterations, the procedure described above halts.*

To obtain an algorithm, we have to find β and K. So far, we have assumed that we already know the Hamiltonian cycles that we aim for. But there is only a polynomial number of possibilities for β and K: For all β, we can assume that for all i there is an edge with $w_i(e) = \beta_i$. Thus, for every i there are at most $O(n^2)$ choices for β_i, hence at most $O(n^{2k})$ in total. The cardinality of K is bounded in terms of $f(k, \varepsilon)$. For fixed k and ε, there is only a polynomial number of subsets of cardinality at most $f(k, \varepsilon)$. We can even restrict ourselves to the subsets K that are path covers: A path cover is a subset K of edges such that K does not contain cycles and both the indegree and outdegree of every vertex is at most one. We obtain MaxATSP-Approx (Alg. 1) and the following theorem.

Theorem 4.3. *For every* $k \geq 1$, $\varepsilon > 0$, MaxATSP-Approx *is a randomized* $\frac{1}{2} - \varepsilon$ *approximation for* k*-criteria Max-ATSP whose running-time for a success probability of at least* $1 - p$ *is polynomial in the input size and* $\log(1/p)$.

Proof. We have to estimate three things: approximation ratio, running-time, and success probability. The proof is by induction on k. For $k = 1$, the theorem holds since there is a deterministic, polynomial-time 2/3 approximation for mono-criterion Max-ATSP. In the following, we assume that the theorem is correct for $k - 1$.

Let us focus on the approximation ratio, the other aspects are omitted for lack of space. For this purpose, we assume that all randomized computations are successful. Let $\tilde{H}$ be an arbitrary tour. For a subset $K \subseteq \tilde{H}$, let $\tilde{H}_K$ be $\tilde{H}$ with all edges in K being contracted. Then, by Lemma 4.2, there exists a (possibly empty) set $K \subseteq \tilde{H}$ of edges of cardinality at most $f(k, \varepsilon/2)$ with one of the two following properties:

(1) There exists an i with $w_i(K) \geq (\frac{1}{2} - \frac{\varepsilon}{2}) \cdot w_i(\tilde{H})$.
(2) For all $e \in \tilde{H}_K$, we have $w(e) \leq (\eta_{k,\varepsilon/2} - (\frac{\varepsilon}{2})^3) \cdot w(\tilde{H}_K)$.

In the first case, there exists an $H' \in \mathcal{P}_{\text{TSP}}^{K,i}$ (see line 13) with $w_j(H') \geq (\frac{1}{2} - \frac{\varepsilon}{2}) \cdot w_j(\tilde{H}_K)$ for all $j \in [k] \setminus \{i\}$. H' combined with K yields a tour H that satisfies $w(H) \geq (\frac{1}{2} - \frac{\varepsilon}{2}) \cdot w(\tilde{H})$: First, we have $w_i(H) \geq w_i(K) \geq \left(\frac{1}{2} - \frac{\varepsilon}{2}\right) \cdot w_i(\tilde{H})$. Second, for $j \neq i$, we have $w_j(H) = w_j(K) + w_j(H') \geq w_j(K) + \left(\frac{1}{2} - \frac{\varepsilon}{2}\right) \cdot w_j(\tilde{H}_K) = \left(\frac{1}{2} - \frac{\varepsilon}{2}\right) \cdot w_j(\tilde{H})$.

In the second case, let $\beta_i = \max\{w_i(e) \mid e \in \tilde{H}_K\} \leq \left(\eta_{k,\varepsilon/2} - (\frac{\varepsilon}{2})^3\right) \cdot w_i(\tilde{H}_K)$. Then $\mathcal{C}_{K,\beta}$ contains a cycle cover C with $w(C) \geq (1 - \frac{\varepsilon}{2}) \cdot w(\tilde{H}_K)$ and $w^\beta(e) \leq \eta_{k,\varepsilon/2} \cdot w(\tilde{H}_K)$ (Lemma 4.1). Thus, C can be decomposed into a collection P of paths with $w(P) \geq (\frac{1}{2} - \frac{\varepsilon}{2}) \cdot w(C)$ (Lemma 4.1). Together with K, this yields a tour H with

$$\begin{aligned} w(H) &\geq w(P) + w(K) \geq \left(\tfrac{1}{2} - \tfrac{\varepsilon}{2}\right) \cdot w(C) + w(K) \geq \left(\tfrac{1}{2} - \tfrac{\varepsilon}{2}\right) \cdot \left(1 - \tfrac{\varepsilon}{2}\right) \cdot w(\tilde{H}_K) + w(K) \\ &= \left(\tfrac{1}{2} - \tfrac{3\varepsilon}{4} + \tfrac{\varepsilon^2}{4}\right) \cdot w(\tilde{H}_K) + w(K) \geq \left(\tfrac{1}{2} - \varepsilon\right) \cdot w(\tilde{H}). \end{aligned}$$

■

4.2. Multi-Criteria Max-STSP

The approximation for MC-Max-ATSP works of course also for MC-Max-STSP. Our goal, however, is a ratio of $(\frac{2}{3} - \varepsilon)$. As a first attempt, one might just replace the $(\frac{1}{2} - \varepsilon)$-decompositions by $(\frac{2}{3} - \varepsilon)$-decompositions. Unfortunately, this is not sufficient since contracting the heavy-weight edges in undirected graphs is not as easy as it is for directed graphs: First, both statements "remove all incoming" and "remove all outgoing" edges are not well-defined in an undirected graph. Second, if we just consider all edges of one vertex as the incoming edges and all edges of the other vertex as the outgoing edges, we obtain a directed graph, which allows only for a ratio of $\frac{1}{2} - \varepsilon$. To circumvent these problems, we do not contract edges $e = \{u, v\}$. Instead, we set the weight of all edges incident to u or v to 0. This allows us to add the edge e to any tour H' without decreasing the weight: We remove all edges incident to u or v from H', and then we add e. The result is a collection of paths. Then we add edges to connect these paths to a Hamiltonian cycle. The only edges that we have removed are edges incident to u or v, which have weight 0 anyway.

However, by setting the weight of edges adjacent to u or v to 0, we might destroy a lot of weight with respect to some objective. To solve this problem as well, we consider larger neighborhoods of the edges in K. In this way, we can add our heavy-weight edge (plus some more edges of its neighborhood) to the Hamiltonian cycle without losing too much weight from removing other edges. The function h in Alg. 2 depends only on k and ε and plays a similar role as f in Section 4.1. We omit the details and obtain MaxSTSP-Approx (Alg. 2) and the following theorem.

Theorem 4.4. *For every $k \geq 1$, $\varepsilon > 0$,* MaxSTSP-Approx *is a randomized $\frac{2}{3} - \varepsilon$ approximation for k-criteria Max-STSP whose running-time for a success probability of at least $1 - p$ is polynomial in the input size and $\log(1/p)$.*

$\mathcal{P}_{\text{TSP}} \leftarrow$ MaxSTSP-Approx$(G, w, k, \varepsilon, p)$
input: undirected complete graph $G = (V, E)$, $k \geq 1$, edge weights $w : E \to \mathbb{N}^k$, $\varepsilon > 0$
output: $(\frac{2}{3} - \varepsilon)$ approximate Pareto curve $\mathcal{P}_{\text{TSP}}$ for k-C-Max-ATSP with a success probability of at least $1 - p$
if $k = 1$ **then**
  compute a 61/81 approximation $\mathcal{P}_{\text{TSP}}$
else
  for all subsets $K \subseteq E$ with $|K| \leq h(k, \varepsilon/3)$ such that K is a path cover **do**
    let L be the set of vertices incident to K
    obtain w^L from w by setting the weight of all edges incident to L to 0
    for all bounds β of (G, w^L) **do**
      $\mathcal{C}_{L,\beta} \leftarrow$ MaxCC-Approx$\big(G, w^{L\beta}, k, \frac{\varepsilon}{3}, \frac{p}{2n^{2k+2h(k,\varepsilon/3)}}\big)$
      for all $C \in \mathcal{C}_{L,\beta}$ with $w^{L\beta}(e) \leq \eta_{k,\varepsilon/3} \cdot w^{L\beta}(C)$ for all $e \in C$ **do**
        $P \leftarrow$ Decompose$(C, w^{L\beta}, \varepsilon/3)$; remove edges of weight 0 from P
        add edges to $K \cup P$ to obtain a tour H; add H to $\mathcal{P}_{\text{TSP}}$
    for all $i \leftarrow 1$ **to** k **do**
      remove the ith objective from w^L to obtain w'^L
      $\mathcal{P}^{L,i}_{\text{TSP}} \leftarrow$ MaxATSP-Approx$(G, w'^L, k-1, \frac{\varepsilon}{3}, \frac{p}{2n^{2k+2h(k,\varepsilon/3)}})$
      for all $H' \in \mathcal{P}^{L,i}_{\text{TSP}}$ **do**
        remove edges of weight 0 from H'
        add edges to $H' \cup K$ to obtain a tour H; add H to $\mathcal{P}_{\text{TSP}}$

Algorithm 2: Approximation algorithm for MC-Max-STSP.

5. Deterministic Approximations for 2-C-Max-STSP

The algorithms presented in the previous section are randomized due to the computation of approximate Pareto curves of cycles covers. So are most approximation algorithms for multi-criteria TSP. As a first step towards deterministic approximation algorithms for MC-Max-TSP, we present a deterministic $61/243 \approx 0.251$ approximation for 2-C-Max-STSP. The key insight for the results of this section is the following lemma, which yields tight bounds for the existence of approximate Pareto curves with only a single element (Theorem 5.2). For completeness, we note that single-element approximate Pareto curves exist for no other variant of multi-criteria TSP than 2-C-Max-STSP.

Lemma 5.1. *Let M be a matching, let H be a collection of paths or a Hamiltonian cycle, and let w be edge weights. Then there exists a subset $P \subseteq H$ such that $P \cup M$ is a collection of paths or a Hamiltonian cycle (we call P in this case an M-feasible set) and $w(P) \geq w(H)/3$.*

Theorem 5.2. *For every undirected complete graph G with edge weights w_1 and w_2, there exists a tour H such that $\{H\}$ is a $1/3$ approximate Pareto curve for 2-C-Max-STSP. This is tight: There exists a graph G with edge weights w_1 and w_2 such that, for all $\varepsilon > 0$, no single Hamiltonian tour of G is a $(1/3 + \varepsilon)$ approximate Pareto curve.*

Lemma 5.1 and Theorem 5.2 are constructive in the sense that, given a tour H_2 that maximizes w_2, the tour H can be computed in polynomial time. A matching M with $w_1(M) \geq w_1(H_1)/3$ can be computed in cubic time. However, since we cannot compute an optimal H_2 efficiently, the results cannot be exploited directly to get an algorithm. Instead, we use an approximation algorithm for finding a tour with as much weight with respect

$\mathcal{P}_{\text{TSP}} \leftarrow$ BiMaxSTSP-Approx(G, w_1, w_2)
input: undirected complete graph $G = (V, E)$, edge weights $w_1, w_2 : E \to \mathbb{N}^k$
output: a 61/243 approximate Pareto curve H
1: compute a maximum-weight matching M with respect to w_1
2: compute a 61/81 approximate tour H_2 with respect to w_2
3: $P \leftarrow H_2 \cap M$; $M' \leftarrow M$; $H_2 \leftarrow H_2 \setminus P$
4: **while** $H_2 \neq \emptyset$ **do**
5: $e \leftarrow \operatorname{argmax}\{w_2(e') \mid e' \in H_2\}$
6: extend e to a path $e_1, \ldots, e_q \in H_2$ such that only e_1 and e_q are incident to edges $z_1, z_2 \in M'$ or the path cannot be extended anymore
7: $P \leftarrow P \cup \{e_1, \ldots, e_q\}$; $H_2 \leftarrow H_2 \setminus \{e_1, \ldots, e_q\}$
8: **if** z_1 or z_2 exists **then**
9: let $f_1, f_2 \in H_2$ be the two edges extending the path if they exist
10: $H_2 \leftarrow H_2 \setminus \{f_1, f_2\}$
11: **if** both z_1 and z_2 exist **then** contract z_1 and z_2 to z; $M' \leftarrow (M' \setminus \{z_1, z_2\}) \cup \{z\}$
12: let H be a tour obtained from $P \cup M$

Algorithm 3: Approximation algorithm for 2-C-Max-STSP.

to w_2 as possible. Using the 61/81 approximation algorithm for Max-STSP [7], we obtain Alg. 3 and the following theorem.

Theorem 5.3. BiMaxSTSP-Approx *is a deterministic* 61/243 *approximation algorithm with running-time* $O(n^3)$ *for* 2*-C-Max-STSP.*

For metric 2-C-Max-STSP, i.e., the edge weights have to fulfil the triangle inequality, we obtain the an approximation ratio of $7/24 > 0.29$ if we replace the 61/81 approximation with the 7/8 approximation for metric Max-STSP by Kowalik and Mucha [15].

6. Approximation Algorithm for MC-Min-ATSP

Now we turn to MC-Min-ATSP and MC-Min-γ-ATSP, i.e., tours of *minimum* weight are sought in directed graphs. Alg. 4 is an adaptation of the algorithm of Frieze et al. [12] to multi-criteria ATSP. Therefore, we briefly describe their algorithm: We compute a cycle cover of minimum weight. If this cycle cover is already a Hamiltonian cycle, then we are done. Otherwise, we choose an arbitrary vertex from every cycle. Then we proceed recursively on the subset of vertices thus chosen to obtain a tour that contains all these vertices. The cycle cover plus this tour form an Eulerian graph. We traverse the Eulerian cycle and take shortcuts whenever we visit vertices more than once. The approximation ratio achieved by this algorithm is $\log_2 n$ for Min-ATSP [12] and $1/(1-\gamma)$ for Min-γ-ATSP [5].

MinATSP-Approx (Alg. 4) for MC-Min-ATSP proceeds as follows: MinCC-Approx computes an approximate Pareto curve of cycle covers. (MinCC-Approx$(G, w, k, \varepsilon, p)$ computes a $(1+\varepsilon)$ approximate Pareto curve of cycle covers of G with weights w with a success probability of at least $1-p$ in time polynomial in the input size, $1/\varepsilon$, and $\log(1/p)$.) Then we iterate by computing approximate Pareto curves of cycle covers on vertex sets V' for every cycle cover C in the previous set. The set V' contains exactly one vertex of every cycle of C. Unfortunately, it can happen that we construct a super-polynomial number of solutions in this way. To cope with this, we remove some intermediate solutions if there are other intermediate solutions whose weight is close by. We call this process *sparsification.*

$\mathcal{P}_{\text{TSP}} \leftarrow$ MinATSP-Approx(G, w, k, ε)
input: directed complete graph $G = (V, E)$ with $n = |V|$, $k \geq 1$, $w : E \to \mathbb{N}^k$, $\varepsilon > 0$
output: $(\log n + \varepsilon)$ approximate Pareto curve for k-C-Min-ATSP or $(\frac{1}{1-\gamma} + \varepsilon)$ approximate Pareto curve for k-C-Min-γ-ATSP with a probability of at least $1/2$

1: $\varepsilon' \leftarrow \varepsilon^2 / \log^3 n$; $\mathcal{F} \leftarrow \emptyset$; $j \leftarrow 1$
2: $\mathcal{C} \leftarrow$ MinCC-Approx$\big(G, w, k, \varepsilon', \frac{1}{2Q \log n}\big)$
3: $\mathcal{P}_0 \leftarrow \{(C, w(C), V, \perp) \mid C \in \mathcal{C}\}$
4: **while** $\mathcal{P}_{j-1} \neq \emptyset$ **do**
5: $\quad \mathcal{P}_j \leftarrow \emptyset$
6: $\quad$ **for all** $\pi = (C', w', V', \pi') \in \mathcal{P}_{j-1}$ **do**
7: $\quad\quad$ **if** (V', C') is connected **then** $\mathcal{F} \leftarrow \mathcal{F} \cup \{(C', w', V', \pi')\}$
8: $\quad\quad$ **else**
9: $\quad\quad\quad$ select one vertex of every component of (V', C') to obtain $\tilde{V}$
10: $\quad\quad\quad \mathcal{C} \leftarrow$ MinCC-Approx$\big(G, w, k, \varepsilon', \frac{1}{2Q \log n}\big)$
11: $\quad\quad\quad \mathcal{P}_j \leftarrow \mathcal{P}_j \cup \{(\tilde{C}, \tilde{w}, \tilde{V}, \pi) \mid \tilde{C} \in \mathcal{C}, \tilde{w} = w' + \gamma^j \cdot w(\tilde{C})\}$
12: $\quad$ **while** there are $\pi', \pi'' \in \mathcal{P}_j$ with equal ε'-signature **do** remove one of them
13: $\quad j \leftarrow j + 1$
14: $\mathcal{P}_{\text{TSP}} \leftarrow \emptyset$
15: **for all** $(C', w', V', \pi') \in \mathcal{F}$ **do**
16: $\quad H \leftarrow C'$
17: $\quad$ **while** $\pi' = (C'', w'', V'', \pi'') \neq \perp$ **do**
18: $\quad\quad$ construct tour H' on V'' from $H \cup C''$ by taking shortcuts such that $H \cap H' = \emptyset$
19: $\quad\quad \pi' \leftarrow \pi''$; $H \leftarrow H'$
20: $\quad \mathcal{P}_{\text{TSP}} \leftarrow \mathcal{P}_{\text{TSP}} \cup \{H\}$

Algorithm 4: Approximation algorithm for MC-Min-ATSP and MC-Min-γ-ATSP.

It is based on the following observation: Let $\varepsilon > 0$, and consider H of weight $w(H) \in \mathbb{N}^k$. For every $i \in \{1, \ldots, k\}$, there is a unique $\ell_i \in \mathbb{N}$ such that $w_i(H) \in [(1+\varepsilon)^{\ell_i}, (1+\varepsilon)^{\ell_i+1})$. We call the vector $\ell = (\ell_1, \ldots, \ell_k)$ the *ε-signature* of H and of $w(H)$. Since $w(H) \leq 2^{p(N)}$, there are at most q^k different ε-signatures for some polynomial k, which is polynomial for fixed k. To get an approximate Pareto curve, we can restrict ourselves to have at most one solution with any specific ε-signature.

In the loop in lines 4 to 13, MinATSP-Approx computes iteratively Pareto curves of cycle covers. The set $\mathcal{P}_j$ contains *configurations* $\pi = (C', w', V', \pi')$, where C' is a cycle cover on V', π' is the predecessor configuration, and w' is the weight of C' plus the weight of its predecessor cycle covers, each weighted with an appropriate power of γ. (We define the ε'-signature of $\pi = (C', w', V', \pi')$ to be the ε'-signature of w'.) These weights are needed for the analysis of the approximation ratio. If, in the course of this computation, we obtain Hamiltonian cycles, these are put into $\mathcal{F}$ (line 7). In line 12, the sparsification takes place. Finally, in lines 14 to 20, Hamiltonian cycles are constructed from the cycle covers computed. In the algorithm, $Q = Q(N, 1/\varepsilon')$ is a two-variable polynomial that bounds the number of different ε'-signatures of solutions for instances of size at most N.

MinATSP-Approx is the first approximation algorithm for MC-Min-ATSP and for MC-Min-γ-ATSP for $\gamma \geq \sqrt{1/3} \approx 0.58$. For $\gamma > 0.55$, it improves over the previously known algorithm [17], which achieves a ratio of $\frac{1}{2} + \frac{\gamma^3}{1-3\gamma^2}$ and works only for $\gamma < \sqrt{1/3} \approx 0.58$.

Theorem 6.1. *For every $\varepsilon > 0$, Alg. 4 is a randomized $(\log n + \varepsilon)$ approximation for MC-Min-ATSP and a randomized $(\frac{1}{1-\gamma} + \varepsilon)$ approximation for MC-Min-γ-ATSP for $\gamma \in [\frac{1}{2}, 1)$. Its running-time is polynomial in the input size and $1/\varepsilon$.*

7. Open Problems

Most approximation algorithms for multi-criteria TSP use randomness for computing approximate Pareto curves of cycle covers. This raises the question if there are algorithms for multi-criteria TSP that are faster, deterministic, and achieve better approximation ratios.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice-Hall, 1993.

[2] Eric Angel, Evripidis Bampis, and Laurent Gourvés. Approximating the Pareto curve with local search for the bicriteria TSP(1,2) problem. *Theoret. Comput. Sci.*, 310(1–3):135–146, 2004.

[3] Eric Angel, Evripidis Bampis, Laurent Gourvès, and Jérǫme Monnot. (Non-)approximability for the multi-criteria TSP(1,2). In *Proc. 15th Int. Symp. on Fundamentals of Computation Theory (FCT)*, vol. 3623 of *LNCS*, pp. 329–340. Springer, 2005.

[4] Markus Bläser, Bodo Manthey, and Oliver Putz. Approximating multi-criteria Max-TSP. In *Proc. 16th Ann. European Symp. on Algorithms (ESA)*, vol. 5193 of *LNCS*, pp. 185–197. Springer, 2008.

[5] Markus Bläser, Bodo Manthey, and Jiří Sgall. An improved approximation algorithm for the asymmetric TSP with strengthened triangle inequality. *J. Discrete Algorithms*, 4(4):623–632, 2006.

[6] L. Sunil Chandran and L. Shankar Ram. On the relationship between ATSP and the cycle cover problem. *Theoret. Comput. Sci.*, 370(1-3):218–228, 2007.

[7] Zhi-Zhong Chen, Yuusuke Okamoto, and Lusheng Wang. Improved deterministic approximation algorithms for Max TSP. *Inform. Process. Lett.*, 95(2):333–342, 2005.

[8] Matthias Ehrgott. Approximation algorithms for combinatorial multicriteria optimization problems. *Int. Trans. Oper. Res.*, 7(1):5–31, 2000.

[9] Matthias Ehrgott. *Multicriteria Optimization*. Springer, 2005.

[10] Matthias Ehrgott and Xavier Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spectrum*, 22(4):425–460, 2000.

[11] Uriel Feige and Mohit Singh. Improved approximation ratios for traveling salesperson tours and paths in directed graphs. In *Proc. 10th Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, vol. 4627 of *LNCS*, pp. 104–118. Springer, 2007.

[12] Alan M. Frieze, Giulia Galbiati, and Francesco Maffioli. On the worst-case performance of some algorithms for the traveling salesman problem. *Networks*, 12(1):23–39, 1982.

[13] Paul C. Gilmore, Eugene L. Lawler, and David B. Shmoys. Well-solved special cases. In Eugene L. Lawler et al., editors, *The Traveling Salesman Problem*, pp. 87–143. John Wiley & Sons, 1985.

[14] Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim I. Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. *J. ACM*, 52(4):602–626, 2005.

[15] Łukasz Kowalik and Marcin Mucha. Deterministic 7/8-approximation for the metric maximum TSP. In *Proc. 11th Int. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, vol. 5171 of *LNCS*, pp. 132–145. Springer, 2008.

[16] Bodo Manthey. On approximating multi-criteria TSP. CoRR 0711.2157 [cs.DS], arXiv, 2008.

[17] Bodo Manthey and L. Shankar Ram. Approximation algorithms for multi-criteria traveling salesman problems. *Algorithmica*, to appear.

[18] Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proc. 41st Ann. IEEE Symp. on Foundations of Computer Science (FOCS)*, pp. 86–92. IEEE Computer Society, 2000.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 649–660
www.stacs-conf.org

TRACTABLE STRUCTURES FOR CONSTRAINT SATISFACTION WITH TRUTH TABLES

DÁNIEL MARX

Department of Computer Science and Information Theory
Budapest University of Technology and Economics
Budapest H-1521, Hungary
E-mail address: dmarx@cs.bme.hu

Abstract. The way the graph structure of the constraints influences the complexity of constraint satisfaction problems (CSP) is well understood for bounded-arity constraints. The situation is less clear if there is no bound on the arities. In this case the answer depends also on how the constraints are represented in the input. We study this question for the truth table representation of constraints. We introduce a new hypergraph measure *adaptive width* and show that CSP with truth tables is polynomial-time solvable if restricted to a class of hypergraphs with bounded adaptive width. Conversely, assuming a conjecture on the complexity of binary CSP, there is no other polynomial-time solvable case.

1. Introduction

Constraint satisfaction is a general framework that includes many standard algorithmic problems such as satisfiability, graph coloring, database queries, etc. A constraint satisfaction problem (CSP) consists of a set V of variables, a domain D, and a set C of constraints, where each constraint is a relation on a subset of the variables. The task is to assign a value from D to each variable such that every constraint is satisfied (see Definition 1.4 for the formal definition). For example, 3SAT can be interpreted as a CSP problem where the domain is $D = \{0, 1\}$ and the constraints in C correspond to the clauses.

In general, solving constraint satisfaction problems is NP-hard if there are no additional restrictions on the instances. The main goal of the research on CSP is to identify tractable special cases of the general problem. The theoretical literature on the CSP investigates two main types of restrictions. The first type is to restrict the *constraint language,* that is, the type of constraints that are allowed. The second type is to restrict the *structure* induced by the constraints on the variables. The *hypergraph* of a CSP instance is defined to be a hypergraph on the variables of the instance such that for each constraint $c \in C$ there is a hyperedge E_c that contains all the variables that appear in c. If the hypergraph of the CSP instance has very simple structure, then the instance is easy to solve. For example, it is

Key words and phrases: computational complexity, constraint satisfaction, treewidth, adaptive width.

Research supported by the Magyary Zoltán Felsőoktatási Közalapítvány and the Hungarian National Research Fund (Grant Number OTKA 67651).

well-known that a CSP instance I with hypergraph H can be solved in time $\|I\|^{O(\mathrm{tw}(H))}$ [5], where $\mathrm{tw}(H)$ denotes the treewidth of H and $\|I\|$ is the size of the representation of I in the input. Thus if we restrict the problem to instances where the treewidth of the hypergraph is bounded by some constant w, then the problem is polynomial-time solvable. The aim of this paper is to investigate whether there exists some other structural property of the hypergraph besides bounded treewidth that makes the problem tractable. Formally, for a class $\mathcal{H}$ of hypergraphs, let $\mathrm{CSP}(\mathcal{H})$ be the restriction of CSP where the hypergraph of the instance is assumed to be in $\mathcal{H}$. Our goal is to characterize the complexity of $\mathrm{CSP}(\mathcal{H})$ for every class $\mathcal{H}$.

We investigate two notions of tractability. $\mathrm{CSP}(\mathcal{H})$ is *polynomial-time solvable* if every instance of $\mathrm{CSP}(\mathcal{H})$ can be solved in time $(\|I\|)^{O(1)}$, where $\|I\|$ is the length of the representation of I in the input. The following notion interprets tractability in a less restrictive way: $\mathrm{CSP}(\mathcal{H})$ is *fixed-parameter tractable (FPT)* if there is a function f such that every instance I of $\mathrm{CSP}(\mathcal{H})$ can be solved in time $f(H)(\|I\|)^{O(1)}$, where H is the hypergraph of the instance. Equivalently, the factor $f(H)$ in the definition can be replaced with a factor $f(k)$ depending only on the number k of vertices of H: as the number of hypergraphs on k vertices is bounded by a function of k, the two definitions result in the same notion. The motivation behind the definition of fixed-parameter tractability is that in certain applications we expect the domain size to be much larger than the number of variables, hence a constant factor in the running time depending only on the number of variables (or on the hypergraph) is acceptable. For more background on fixed-parameter tractability, see [3, 4].

Bounded arities. If the constraints have bounded arity (i.e., edge size in $\mathcal{H}$ is bounded by a constant), then the complexity of $\mathrm{CSP}(\mathcal{H})$ is well understood:

Theorem 1.1 ([7]). *If $\mathcal{H}$ is a recursively enumerable class of hypergraphs with bounded edge size, then (assuming* FPT $\neq$ W[1]*) the following are equivalent:*

(1) $\mathrm{CSP}(\mathcal{H})$ *is polynomial-time solvable.*
(2) $\mathrm{CSP}(\mathcal{H})$ *is fixed-parameter tractable.*
(3) $\mathcal{H}$ *has bounded treewidth.*

The assumption FPT $\neq$ W[1] is a standard hypothesis of parameterized complexity. Thus in the bounded arity case bounded treewidth is the only property of the hypergraph that can make the problem polynomial-time solvable. Furthermore, the following sharpening of Theorem 1.1 shows that there is no algorithm whose running time is significantly better than the $\|I\|^{O(\mathrm{tw}(H))}$ bound of the treewidth based algorithm. The result is proved under the Exponential Time Hypothesis (ETH) [9]: it is assumed that there is no $2^{o(n)}$ time algorithm for n-variable 3SAT.

Theorem 1.2 ([11]). *If there is a computable function f and a recursively enumerable class $\mathcal{H}$ of hypergraphs with bounded edge size and unbounded treewidth such that the problem* $\mathrm{CSP}(\mathcal{H})$ *can be solved in time $f(H)\|I\|^{o(\mathrm{tw}(H)/\log \mathrm{tw}(H))}$ for instances I with hypergraph $H \in \mathcal{H}$, then ETH fails.*

This means that the treewidth-based algorithm is almost optimal: in the exponent only an $O(\log \mathrm{tw}(H))$ factor improvement is possible. It is conjectured in [11] that Theorem 1.2 can be made tight:

Conjecture 1.3 ([11]). *If $\mathcal{H}$ is a class of hypergraphs with bounded edge size, then there is no algorithm that solves* $\mathrm{CSP}(\mathcal{H})$ *in time $f(H)\|I\|^{o(\mathrm{tw}(H))}$ for instances I with hypergraph $H \in \mathcal{H}$, where f is an arbitrary computable function.*

Unbounded arities. The situation is less understood in the unbounded arity case, i.e., when there is no bound on the maximum edge size in $\mathcal{H}$. First, the complexity in the unbounded-arity case depends on how the constraints are represented. In the bounded-arity case, if each constraint contains at most r variables (r being a fixed constant), then every reasonable representation of a constraint has size $|D|^{O(r)}$. Therefore, the size of the different representations can differ only by a polynomial factor. On the other hand, if there is no bound on the arity, then there can be exponential difference between the size of succinct representations (e.g., formulas) and verbose representations (e.g., truth tables). The running time is expressed as a function of the input size, hence the complexity of the problem can depend on how the input is represented: longer representation means that it can be easier to obtain a polynomial-time algorithm.

The most well-studied representation of constraints is listing all the tuples that satisfy the constraint. For this representation, there are classes $\mathcal{H}$ with unbounded treewidth such that CSP restricted to this class is polynomial-time solvable. For example, classes with bounded *(generalized) hypertree width* [6], bounded *fractional edge cover number* [8], and bounded *fractional hypertree width* [8, 10] are such classes. However, no classification theorem similar to Theorem 1.1 is known for this version. More succinct representations were studied by Chen and Grohe [2]: constraints are represented by generalized DNF formulas and ordered binary decision diagrams (OBDD). The complexity of the problem with this representation was fully characterized: the complexity depends not on the treewidth of the hypergraph, but on the treewidth of the incidence structure.

Truth table representation. In this paper we study another natural representation: truth tables. A constraint of arity r is represented by having one bit for each possible r-tuple that can appear on the r variables of the constraint, and this bit determines whether this particular r-tuple satisfies the constraint or not. To increase the flexibility of the representation and make it more natural, we allow that the variables have different domains, i.e., each variable v has to be assigned a value from its domain $\mathrm{Dom}(v)$. Thus the size of the truth table of an r-ary constraint is proportional to the size of the direct product of the domains of the r variables. This representation is more verbose than listing satisfying tuples: the size of the representation is proportional to the number of possible tuples even if only few tuples satisfy the constraint. We define CSP as follows:

Definition 1.4. A *CSP instance* is a quadruple (V, D, Dom, C), where:

- V is a set of variables,
- D is a domain of values,
- $\mathrm{Dom} : V \to 2^D$ assigns a domain $\mathrm{Dom}(v) \subseteq D$ to each variable $v \in V$,
- C is a set of constraints. Each $c_i \in C$ is a pair $\langle s_i, R_i \rangle$, where:
  - $s_i = (u_{i,1}, \ldots, u_{i,m_i})$ is a tuple of variables (the *constraint scope*), and
  - R_i is a subset of $\prod_{j=1}^{m_i} \mathrm{Dom}(u_{i,j})$ (the *constraint relation*).

For each constraint $\langle s_i, R_i \rangle$ the tuples of R_i indicate the allowed combinations of values for the variables in s_i. The length m_i of the tuple s_i is called the *arity* of the constraint. A *solution* to a CSP instance is a function $f : V \to D$ such that $f(v) \in \mathrm{Dom}(v)$ for every $v \in V$ and for each constraint $\langle s_i, R_i \rangle$ with $s_i = \langle u_{i,1}, u_{i,2}, \ldots, u_{i,m_i} \rangle$, the tuple $\langle f(u_{i,1}), f(u_{i,2}), \ldots, f(u_{i,m_i}) \rangle$ is in R_i.

We denote by $\mathrm{CSP}_{\mathrm{tt}}$ the problem where each constraint $\langle s_i, R_i \rangle$ of arity m_i is represented by the truth table of the constraint relation R_i, that is, by a sequence of $\prod_{j=1}^{m_i} |\mathrm{Dom}(u_{i,j})|$

bits that describe that subset R_i of $\prod_{j=1}^{m_i} \mathrm{Dom}(u_{i,j})$. For a class $\mathcal{H}$, $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ is the restriction to instances with hypergraph in $\mathcal{H}$.

Results. The main result of the paper is a complete characterization of the complexity of $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ (assuming Conjecture 1.3). The complexity of the problem depends on a new hypergraph measure *adaptive width:*

Theorem 1.5 (Main)**.** *Assuming Conjecture 1.3, the following are equivalent:*

(1) $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ *is polynomial-time solvable.*
(2) $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ *is fixed-parameter tractable.*
(3) $\mathcal{H}$ *has bounded adaptive width.*

The assumption in Theorem 1.5 is nonstandard, so it is up to the reader to decide how strong this evidence is. However, the message of Theorem 1.5 is the following: a new tractable class for $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ would imply surprising new results for *binary* CSP. Thus it is not worth putting too much effort in further studying $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ with the hope of finding new tractable classes: as this would disprove Conjecture 1.3, such an effort would be better spent trying to disprove Conjecture 1.3 directly, by beating the $\|I\|^{O(\mathrm{tw}(H))}$ algorithm for binary CSP.

Listing the satisfying tuples is a more succinct representation of a constraint than a truth table. Thus if CSP is polynomial-time solvable or fixed-parameter tractable for some class $\mathcal{H}$ with the former representation, then this also holds for the latter representation as well. In particular, this means that by the results of [8, 10], $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ is polynomial-time solvable if $\mathcal{H}$ has bounded fractional hypertree width. This raises the question whether Theorem 1.5 gives any new tractable class $\mathcal{H}$. In other words, is there a class $\mathcal{H}$ having bounded adaptive width but unbounded fractional hypertree width? In Section 5, we answer this question by constructing such a class $\mathcal{H}$. This means that $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ is polynomial-time solvable, but if the constraints are represented by listing the satisfying tuples, then it is not even known whether the problem is FPT.

2. Width parameters

Treewidth and various variants are defined in this section. We follow the framework of width functions introduced by Adler [1]. A *tree decomposition* of a hypergraph H is a tuple $(T, (B_t)_{t\in V(T)})$, where T is a tree and $(B_t)_{t\in V(T)}$ is a family of subsets of $V(H)$ such that for each $E \in E(H)$ there is a node $t \in V(T)$ such that $E \subseteq B_t$, and for each $v \in V(H)$ the set $\{t \in V(T) \mid v \in B_t\}$ is connected in T. The sets B_t are called the *bags* of the decomposition. Let $f : 2^{V(H)} \to \mathbb{R}^+$ be a function that assigns a real number to each subset of vertices. The *f-width* of a tree-decomposition $(T, (B_t)_{t\in V(T)})$ is $\max\{f(B_t) \mid t \in V(T)\}$. The *$f$-width* of a hypergraph H is the minimum of the f-widths of all its tree decompositions.

Definition 2.1. Let $s(B) = |B| - 1$. The *treewidth* of H is $\mathrm{tw}(H) := s\text{-width}(H)$.

A subset $E' \subseteq E(H)$ is an *edge cover* if $\bigcup E' = V(H)$. The *edge cover number* $\rho(H)$ is the size of the smallest edge cover (assuming H has no isolated vertices). For $X \subseteq V(H)$, let $\rho_H(X)$ be the size of the smallest set of edges covering X.

Definition 2.2. The *(generalized) hypertree width* of H is $\mathrm{hw}(H) := \rho_H\text{-width}(H)$.

We also consider the linear relaxations of edge covers: a function $\gamma : E(H) \to [0, 1]$ is a *fractional edge cover* of H if $\sum_{E:v\in E} \gamma(E) \geq 1$ for every $v \in V(H)$. The *fractional*

cover number $\rho^*(H)$ of H is the minimum of $\sum_{E\in E(H)} \gamma(E)$ taken over all fractional edge covers of H. We define $\rho^*_H(X)$ analogously to $\rho_H(X)$: the requirement $\sum_{E:v\in E} \gamma(E) \geq 1$ is restricted to vertices of X.

Definition 2.3. The *fractional hypertree width* of H is $\mathrm{fhw}(H) := \rho^*_H\text{-width}(H)$.

The dual of covering is independence. A subset $X \subseteq V(H)$ is an *independent set* if $|X \cap E| \leq 1$ for every $E \in E(H)$. The *independence number* $\alpha(H)$ is the size of the largest independent set and $\alpha_H(X)$ is the size of the largest independent set that is a subset of X. A function $\phi : V(H) \to [0,1]$ is a *fractional independent set* of the hypergraph H if $\sum_{v\in E} \phi(v) \leq 1$ for every $E \in E(H)$. The *fractional independence number* $\alpha^*(H)$ of H is the maximum of $\sum_{v\in V(H)} \phi(v)$ taken over all fractional independent sets ϕ of H. It is well-known that $\alpha(H) \leq \alpha^*(H) = \rho^*(H) \leq \rho(H)$ for every hypergraph H. Thus α^*-width gives us the same notion as fractional hypertree width. The main new definition of the paper uses fractional independent sets, but in a different way. For a function $f : V(H) \to \mathbb{R}^+$, we define $f(X) = \sum_{v\in X} f(v)$ for $X \subseteq V(H)$ and define f-width accordingly.

Definition 2.4. The *adaptive width* $\mathrm{adw}(H)$ of a hypergraph H is the maximum of ϕ-width(H) taken over all fractional independent sets ϕ of H.

Currently, we do not have an efficient algorithm for computing adaptive width. Fortunately, the polynomial-time algorithm in Section 3 for instances with bounded adaptive width does not need to determine the adaptive width of the input, it is sufficient that the adaptive width is promised to be bounded. However, the hardness proof of Section 4 requires that the question $\mathrm{adw}(H) \geq w$ is decidable (proof is omitted).

Lemma 2.5. *There is an algorithm that, given hypergraph H and rational number w, decides if* $\mathrm{adw}(H) \geq w$. *If the answer is yes, then the algorithm returns a rational fractional independent set α such that the α-width of H is at least w.*

We finish the section with a combinatorial observation (the closed neighborhood of a vertex v is the union of all the edges containing v):

Lemma 2.6. *Given a tree decomposition of hypergraph H, it can be transformed in polynomial time (by removing vertices from some bags) into a tree decomposition of H satisfying the following property: if two adjacent vertices u and v have the same closed neighborhood, then u and v appear in exactly the same bags.*

Proof. Consider a tree decomposition of H. If u and v are two vertices that do not satisfy the requirements, then remove these vertices from those bags where only one of them appears (since u and v are neighbors, they appear together in some bag B_t, hence both vertices appear in at least one bag after the removals). The intersection of two subtrees is also a subtree, thus it remains true that u and v appear in a connected subset of the bags. We have to show that for every edge $E \in E(H)$, there is a bag B_t that fully contains E even after the removals. If $\{u,v\} \subseteq E$ or $E \cap \{u,v\} = \emptyset$, then this clearly follows from that fact that some bag fully contains E before the removals. Assume without loss of generality that $u \in E$ and $v \notin E$. We show that $E \cup \{v\}$ is fully contained in some bag B_t before the removals, hence (as $\{u,v\} \subseteq E \cup \{v\}$) B_t fully contains $E \cup \{v\}$ even after the removals. Since $u \in E$, edge E is in the closed neighborhood of v. Thus by assumption, E is also in the closed neighborhood of v, which means that $E \cup \{v\}$ is a clique in H. It is well known that every clique is fully contained in some bag of the tree decomposition (this follows from

the fact that subtrees of a tree satisfy the Helly property), thus it follows that $E \cup \{v\} \subseteq B_t$ for some bag B_t.

Let us repeat these removals until there are no pairs u, v that violate the requirements; eventually we get a tree decomposition as required. Observe that the procedure terminates after a polynomial number of steps: vertices are only removed from the bags. ∎

3. Algorithm for bounded adaptive width

We prove that $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ is polynomial-time solvable if $\mathcal{H}$ has bounded adaptive width. Bounded adaptive width ensures that no matter what the distribution of the domain sizes in the input instance is, there is a decomposition where the variables in each bag have only a polynomial number of possible assignments. For such a decomposition, the instance can be solved by standard techniques.

Lemma 3.1. *There is an algorithm that, given an instance I of $\mathrm{CSP}_{\mathrm{tt}}$, an integer C, and a tree decomposition $(T, (B_t)_{t \in V(T)})$ of the hypergraph H of the instance such that $\prod_{v \in B_t} |\operatorname{Dom}(v)| \leq C$ for every bag B_t, solves the instance I in time polynomial in $\|I\| \cdot C$.*

Proof. If $\prod_{v \in B_t} |\operatorname{Dom}(v)| \leq C$, then there are at most C possible assignments on the variables in B_t. Using standard dynamic programming techniques, it is easy to check whether it is possible to select one assignment f_t for each bag B_t such that f_t satisfies the instance induced by the bag B_t and these assignments are compatible. For completeness, we briefly describe how this can be done by a reduction to binary CSP.

Let us construct a binary CSP instance I' as follows. The set of variables of I' is $V(T)$, i.e., the bags of the tree decomposition. For $t \in V(T)$, let $b_t \leq C$ be the number of assignments f to the variables in B_t such that $f(v) \in \operatorname{Dom}(v)$ for every $v \in B_t$; denote by $f_{t,i}$ the i-th such assignment on B_t ($1 \leq i \leq b_t$). The domain of I' is $D' = \{1, \ldots, C\}$. For each edge $t't'' \in E(T)$, we introduce a constraint $c_{t',t''} = \langle (t', t''), R_{t',t''} \rangle$, where $(i, j) \in R_{t',t''}$ if and only if

- $f_{t',i}$ and $f_{t'',j}$ are compatible, i.e., $f_{t',i}(v) = f_{t'',j}(v)$ for every $v \in B_{t'} \cap B_{t''}$.
- $f_{t',i}$ satisfies every constraints of I whose scope is contained in $B_{t'}$.
- $f_{t'',j}$ satisfies every constraints of I whose scope is contained in $B_{t''}$.

It is easy to see that a solution of I' determines a solution of I. The size of I' is polynomial in C and $\|I\|$. Since the graph of I' is a tree, it can be solved in time $\|I'\|^{O(1)} = (\|I\|C)^{O(1)}$. ∎

Theorem 3.2. *If $\mathcal{H}$ has bounded adaptive width, then $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H}) \in$ PTIME.*

Proof. Let I be an instance of $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ with hypergraph H such that $\operatorname{adw}(H) \leq c$. Let $N \leq \|I\|$ be the size of the largest truth table in the input; we assume that $N > 1$, since the problem is trivial if $N = 1$. We show that it is possible to find in time $N^{O(c)}$ a tree decomposition $(T, B_{t \in V(T)})$ of the instance such that $\prod_{v \in B_t} |\operatorname{Dom}(v)| \leq N^{O(c)}$ holds for every bag B_t. By Lemma 3.1, this means that the instance can be solved in time polynomial in $\|I\|$ and $N^{O(c)}$, i.e., the running time is $\|I\|^{O(c)}$.

Let $\phi(v) = \log_2 |\operatorname{Dom}(v)| / \log_2 N$. We claim that ϕ is a fractional independent set of H. If there is a constraint with $(v_{i_1}, v_{i_2}, \ldots, v_{i_r})$ such that $\sum_{j=1}^{r} \phi(v_j) > 1$, then the size of the truth table describing the constraint is larger than N:

$$\prod_{j=1}^{r} |\operatorname{Dom}(i_j)| = \prod_{j=1}^{r} 2^{\phi(v_{i_j}) \cdot \log_2 N} = 2^{\log_2 N \cdot \sum_{j=1}^{r} \phi(v_{i_j})} > 2^{\log_2 N} = N.$$

Define $\phi'(v) = \lceil \phi(v) \log_2 N \rceil$. Observe that $\phi(v) \geq 1/\log_2 N$, hence $\phi'(v) < 2\phi(v)\log_2 N$ (if $|\operatorname{Dom}(v)| = 1$, then the instance can be simplified). Let H' be the hypergraph that is obtained from H by replacing each vertex v with a set X_v of $\phi'(v)$ vertices; if an edge E contains some vertex v in H, then E contains every vertex of X_v in H'. We claim that H' has treewidth less than $2c\log_2 N$. Since $\operatorname{adw}(H) \leq c$, H has a tree decomposition $(T, B_{t\in V(T)})$ such that $\sum_{v\in B_t} \phi(v) \leq c$ holds for every bag B_t. Consider the analogous decomposition $(T, B'_{t\in V(T)})$ of H'; i.e., if a bag B_t contains a vertex v of H, then let bag B'_t contain every vertex of X_v. The size of a bag B'_t is $\sum_{v\in B_t} |X_v| = \sum_{v\in B_t} \phi'(v) \leq 2\log_2 N \cdot \sum_{v\in B_t} \phi(v) \leq 2c\log_2 N$, thus the treewidth of H' is indeed less than $2c\log_2 N$. Given a graph G with n vertices, it is possible to find a tree decomposition of width at most $4\operatorname{tw}(G)+1$ in time $2^{O(\operatorname{tw}(G))}n^{O(1)}$ (see e.g., [4, Prop. 11.14]). Thus we can a find a tree decomposition $(T, B''_{t\in V(T)})$ of width at most $8c\log_2 N$ for H' in time $2^{O(2c\log_2 N)}||H'||^{O(1)} = N^{O(c)}||H'||^{O(1)}$.

In H', every vertex of X_v is contained in the same set of edges. By Lemma 2.6, it can be assumed that each bag of $(T, B''_{t\in V(T)})$ contains either all or none of X_v. Define the tree decomposition $(T, B^*_{t\in V(T)})$ of H where bag B^*_t contains v if and only X_v is contained in B''_t. The ϕ-weight of a bag B^*_t can be bounded as

$$\sum_{v\in B^*_t} \phi(v) \leq \frac{1}{\log_2 N} \sum_{v\in B^*_t} \phi'(v) = \frac{1}{\log_2 N}|B''_t| \leq 8c.$$

Thus in the tree decomposition $(T, B^*_{t\in V(T)})$, the product of the domain sizes is

$$\prod_{v\in B^*_t} |\operatorname{Dom}(v)| = \prod_{v\in B^*_t} 2^{\phi(v)\cdot\log_2 N} = 2^{\log_2 N\cdot\sum_{v\in B^*_t}\phi(v)} \leq 2^{\log_2 N\cdot 8c} = N^{8c},$$

in each bag B^*_t, as required. ∎

4. Hardness result for unbounded adaptive width

We prove the main complexity result of the paper in this section.

Theorem 4.1. *Let $\mathcal{H}$ be a recursively enumerable class of hypergraphs with unbounded adaptive width. Assuming Conjecture 1.3,* $\operatorname{CSP}_{\text{tt}}(\mathcal{H})$ *is not FPT.*

Proof. Suppose that $\operatorname{CSP}_{\text{tt}}(\mathcal{H})$ can be solved in time $h_1(H)\|I\|^c$ for some constant c and computable function h_1. Let us fix an arbitrary computable enumeration of the hypergraphs in $\mathcal{H}$. For every $k \geq 1$, let H_k be the first hypergraph in this enumeration with $\operatorname{adw}(H_k) \geq k$. For each $k \geq 1$, let ϕ_k be the fractional independent set returned by the algorithm of Lemma 2.5 for the question '$\operatorname{adw}(H_k) \geq k$?'.

Constructing the graph class $\mathcal{G}$. For each $k \geq 1$, we construct a graph G_k based on H_k and ϕ_k. Let q_k be the least common denominator of the rational values $\phi_k(v)$ for $v \in V(H_k)$. The graph G_k has a clique K_v of size $q_k \cdot \phi(v)$ for each $v \in V(H_k)$ and if u and v are neighbors in H_k, then every vertex of K_u is connected to every vertex of K_v. Let $\mathcal{G} = \{G_k \mid k \geq 1\}$.

We claim that $\operatorname{tw}(G_k) \geq q_k k - 1$. Suppose for contradiction that G_k has a tree decomposition $(T, (B_t)_{t\in V(T)})$ of width less than $q_k k - 1$, i.e., the size of every bag is smaller than $q_k k$. By Lemma 2.6, it can be assumed that for every $v \in V(H_k)$ and bag B_t of

the decomposition, either B_t fully contains K_v or disjoint from it. Let us construct a tree decomposition $(T, (B'_t)_{t \in V(T)})$ of H_k such that B'_t contains v if and only if B_t fully contains K_v. It is easy to see that this is a tree decomposition of H_k: for every $E \in E(H_k)$, the set $\bigcup_{v \in E} K_v$ is a clique in G_k, hence there is a bag B_t containing $\bigcup_{v \in E} K_v$, i.e, B'_t contains E. Furthermore, $\phi_k(B'_t) < k$ for every bag B'_t: if $\phi_k(B'_t) \geq k$, then $|\bigcup_{v \in E} K_v| \geq q_k k$, contradicting the assumption that every bag B_t has size strictly less than $q_k k$. This would contradict the assumption $\operatorname{adw}(H_k) \geq k$, thus $\operatorname{tw}(G_k) \geq q_k k - 1$.

Simulating G_k by H_k. We present an algorithm for $\#\mathrm{CSP}(\mathcal{G})$ violating Conjecture 1.3. We show how a binary $\#\mathrm{CSP}(\mathcal{G})$ instance I_1 with graph G_k can be reduced to a $\#\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ instance I_2 with hypergraph $H_k \in \mathcal{H}$. Then I_2 can be solved with the assumed algorithm for $\#\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$. Let $G \in \mathcal{G}$ be the graph of the CSP instance I_1. By enumerating the hypergraphs in $\mathcal{H}$, we can find the first value k such that $G = G_k$. We construct a $\#\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ instance I_2 with hypergraph H_k where every variable $v \in V(H_k)$ simulates the variables in K_v.

The domain $\operatorname{Dom}(v)$ of v is $D^{|K_v|}$, i.e., $\operatorname{Dom}(v)$ is the set of $|K_v|$-tuples of D. For every $v \in V(H_k)$, there is a natural bijection between the elements of $\operatorname{Dom}(v)$ and the $|D|^{|K_v|}$ possible assignments $f : K_v \to D$. For each edge $E = (v_1, \ldots, v_r) \in E(H_k)$, we add a constraint $c_E = \langle (v_1, \ldots, v_r), R_E \rangle$ to I_2 as follows. Let $(x_1, \ldots, x_r) \in \prod_{i=1}^{r} \operatorname{Dom}(v_i)$. For $1 \leq i \leq r$, let g_i be the assignment of K_{v_i} corresponding to $x_i \in \operatorname{Dom}(v_i)$. These r assignments together define an assignment $g : \bigcup_{i=1}^{r} K_{v_i} \to D$ on the union of their domains. We define the relation R_E such that $(x_1, \ldots, x_r)$ is a member of R_E if and only if the corresponding assignment g satisfies every constraint of I_1 whose scope is contained in $\bigcup_{i=1}^{r} K_{v_i}$.

Assume that I_1 has a solution $f_1 : V(G_k) \to D$. For every $v \in V(H_k)$, define $f_2(v)$ to be the member of $\operatorname{Dom}(v)$ corresponding to the assignment f_1 restricted to K_v. Now f_2 is a solution of I_2: for every edge E of H_k, assignment f_1 restricted to $\bigcup_{v \in E} K_v$ clearly satisfies every constraint of I_1 whose scope is in $\bigcup_{v \in E} K_v$.

Assume now that I_2 has a solution $f_2 : V_2 \to D_2$. For every $v \in V(H_k)$, there is an assignment $f_v : K_v \to D$ corresponding to the value $f_2(v)$. These assignments together define an assignment $f_1 : V(G_k) \to D$. We claim that f_1 is a solution of I_1. Let $c = \langle (u', v'), R \rangle$ be an arbitrary constraint of I_1. Assume that $u' \in K_u$ and $v' \in K_v$ for some $u, v \in V(H_k)$. Since $u'v' \in E(G_k)$, there is an edge $E \in E(H_k)$ with $u, v \in E$. The definition of c_E in I_2 ensures that f_1 restricted to $K_u \cup K_v$ satisfies every constraint of I_1 whose scope is contained in $K_u \cup K_v$; in particular, f_1 satisfies constraint c.

Running time. Assume that an instance I_1 of $\#\mathrm{CSP}(\mathcal{G})$ is solved by first reducing it to an instance I_2 as above and then applying the algorithm for $\#\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$. Let us determine the running time of this algorithm. The first step of the algorithm is to enumerate the hypergraphs in $\mathcal{H}$ until the correct value of k is found. The time required by this step depends only on the graph $G \in \mathcal{G}$; denote it by $h_2(G)$. Let us determine the time required to construct instance I_2 and the size of the representation of I_2. As defined above, for each constraint c_E in I_2, we have to enumerate every tuple $(x_1, \ldots, x_r) \in \prod_{i=1}^{r} \operatorname{Dom}(v)$ and check whether the corresponding assignment g is a solution of the instance $I_1[\bigcup_{i=1}^{r} K_{v_i}]$. Checking a vector $(x_1, \ldots, x_r)$ can be done in time polynomial in $\|I_1\|$. Moreover,

$$\left| \prod_{i=1}^{r} \operatorname{Dom}(v) \right| = \prod_{i=1}^{r} |D|^{|K_v|} = \prod_{i=1}^{r} |D|^{q_k \cdot \phi_k(v)} = |D|^{q_k \sum_{i=1}^{r} \phi_k(v)} \leq |D|^{q_k},$$

since ϕ_k is a fractional independent set and $\{x_1, \ldots, x_r\}$ is an edge of H_k. Every other step is polynomial in $\|I_1\|$, hence the reduction can be done in time $h_2(G)\|I_1\|^{O(q_k)}$, which is also a bound on $\|I_2\|$. Thus the algorithm for $\mathrm{CSP}_{\mathrm{tt}}(\mathcal{H})$ requires $h_1(H_k)(h_2(G)\|I_1\|)^{O(q_k c)}$ time, yielding a total time of $h_3(G)\|I_1\|^{O(q_k c)}$ for some computable function h_3.

We show that $\|I_1\|^{O(q_k c)}$ is $\|I_1\|^{o(\mathrm{tw}(G_k))}$, violating Conjecture 1.3. Let $s(w)$ be the smallest k such that $\mathrm{tw}(G_k)$ is greater than w (as $\mathrm{tw}(G_k) \geq q_k k - 1$, this is well defined). Observe that $s(w)$ is nondecreasing and unbounded. We have

$$\|I_1\|^{O(q_k c)} \leq \|I_1\|^{O(c(\mathrm{tw}(G_k)+1)/k)} \leq \|I_1\|^{O(c(\mathrm{tw}(G)+1)/s(\mathrm{tw}(G)))} = \|I_1\|^{o(\mathrm{tw}(G))}.$$

Thus the total running time is $h_3(G)\|I_1\|^{o(\mathrm{tw}(G))}$, violating Conjecture 1.3. ∎

5. Relation of bounded fractional hypertree width and bounded adaptive width

We show that the class of sets of hypergraphs with bounded adaptive width strictly includes the class of sets with bounded fractional hypertree width. First, fractional hypertree width is an upper bound for adaptive width.

Proposition 5.1. *For every hypergraph H,* $\mathrm{adw}(H) \leq \mathrm{fhw}(H)$.

Proof. Let $(T, B_{t \in V(T)})$ be a tree decomposition of H whose ρ^*_H-width is $\mathrm{fhw}(H)$. If ϕ is a fractional independent set, then $\phi(B_t) \leq \rho^*_H(B_t) \leq \mathrm{fhw}(H)$ for every bag B_t of the decomposition, i.e., ϕ-width$(H) \leq \mathrm{fhw}(H)$. This is true for every fractional independent set ϕ, hence $\mathrm{adw}(H) \leq \mathrm{fhw}(H)$. ∎

This implies that if a set of hypergraphs has bounded fractional hypertree width, then it has bounded adaptive width as well. The converse is not true: the main result of this section is a set of hypergraphs with bounded adaptive width (Corollary 5.11) that has unbounded fractional hypertree width (Corollary 5.8).

Definition 5.2. The hypergraph $H(d, c)$ has $2^{d+1} - 1$ vertices $v_{i,j}$ $(0 \leq i \leq d,\ 0 \leq j < 2^i)$ and the following edges:

- For every $0 \leq k < 2^d$, there is a *large edge* E_k of size $d + 1$ that contains $v_{i,\lfloor k/2^{d-i} \rfloor}$ for every $0 \leq i \leq d$.
- For every i, j_1, j_2 with $|j_1 - j_2| \leq c$, there is a *small edge* $\{v_{i,j_1}, v_{i,j_2}\}$.

We say that vertex $v_{i,j}$ is on *level* i. We define $\chi(v_{i,j}) = j2^{d-i}$. The set $\mathcal{H}_c$ contains every hypergraph $H(d, c)$ for $d \geq 1$.

Definition 5.3. If $v_{i,j}$ and $v_{i',j'}$ are covered by the same large edge E_k and $i \leq i'$, then $v_{i,j}$ is an *ancestor* of $v_{i',j'}$; and $v_{i',j'}$ is a *descendant* of $v_{i,j}$.

Proposition 5.4. *If $v_{i,j}$ is an ancestor of $v_{i',j'}$, then $\chi(v_{i,j}) \leq \chi(v_{i',j'}) < \chi(v_{i,j}) + 2^{d-i}$.*

Proof. The ancestor of $v_{i',j'}$ on level i is $v_{i,\lfloor j'/2^{i'-i} \rfloor}$. Therefore,

$$\chi(v_{i,j}) = \lfloor j'/2^{i'-i} \rfloor \cdot 2^{d-i} \leq j'/2^{d-i'} = \chi(v_{i',j'})$$

and

$$\chi(v_{i,j}) = \lfloor j'/2^{i'-i} \rfloor \cdot 2^{d-i} > (j'/2^{i'-i} - 1) \cdot 2^{d-i} = j' \cdot 2^{d-i'} - 2^{d-i} = \chi(v_{i',j'}) - 2^{d-i}.$$

∎

5.1. Lower bound on fractional hypertree width

Fractional hypertree width has various other characterizations that are equivalent up to a constant factor [8]. Here we use the characterization by balanced separators to prove a lower bound on the fractional hypertree width of $H(d,c)$.

For a function $\gamma : E(H) \to \mathbb{R}^+$, we define $\text{weight}(\gamma) := \sum_{E \in E(H)} \gamma(E)$. For a set $W \subseteq V(H)$, we let $\text{weight}(\gamma|W) = \sum_{e \in E_W} \gamma(e)$, where E_W is the set of all edges intersecting W. For $\lambda > 0$, a set $S \subseteq V(H)$ is a *λ-balanced separator for γ* if $\text{weight}(\gamma|C) \leq \lambda \cdot \text{weight}(\gamma)$ for every component C of $H \setminus S$.

Theorem 5.5 ([8])**.** *Let H be a hypergraph and $\gamma : E(H) \to \mathbb{R}^+$. There is a $\frac{1}{2}$-balanced separator S for γ such that $\rho^*_H(S) \leq \text{fhw}(H)$.*

Theorem 5.5 can be generalized to λ-separators with arbitrary $\lambda > 0$ (proof is omitted):

Corollary 5.6. *Let H be a hypergraph and $\gamma : E(H) \to \mathbb{R}^+$. For every $\lambda > 0$, there is a λ-balanced separator S for γ such that $\rho^*_H(S) \leq 2\,\text{fhw}(H)/\lambda$.*

Proposition 5.7. *For every $c \geq 5$ and $d > 2\log_2 c$, $\text{fhw}(H(d,c)) \geq \sqrt{d}/(2c)$.*

Proof. Let γ be a weight function on the edges that assigns 1 to each large edge and 0 to the small edges. We show that every $\frac{1}{2c}$-balanced separator of $H(d,c)$ for γ has fractional cover number at least $\sqrt{d}/2$. By Corollary 5.6, $\text{fhw}(H(d,c)) \geq \sqrt{d}/(8c)$ follows.

Suppose that S is a $\frac{1}{2c}$-balanced separator of $H(d,c)$ for γ. Observe that on level $d/2$, there are at least c vertices: $2^{d/2} \geq c$. We claim that there is a $d/2 \leq i \leq d$ for which there is no $0 \leq a_i \leq 2^i - c$ such that $v_{i,j} \in S$ for every $a_i \leq j < a_i + c$. Suppose that there is such an a_i for every $d/2 \leq i \leq d$. Let $b_i = a_i + c - 1$. It follows from the definition of a_i that $v_{i,a_i}, v_{i,b_i} \in S$ for every $d/2 \leq i \leq d$. We claim that the set $X = \{v_{i,a_i}, v_{i,b_i} : d/2 \leq i \leq d\}$ contains an independent set of size at least $\sqrt{d}/2$, contradicting the assumption that the fractional cover number $\rho^*(S)$ is less than $\sqrt{d}/2$ (recall that $\alpha_H(S) \leq \rho^*_H(S)$ holds). First we show that if a large edge E_k covers v_{i,a_i} and $v_{i',a_{i'}}$ then v_{i,b_i} and $v_{i',b_{i'}}$ are independent. Assume without loss of generality that $i < i'$. By Prop. 5.4, $|\chi(v_{i,a_i}) - \chi(v_{i',a_{i'}})| < 2^{d-i}$. Since $\chi(v_{i,b_i}) = \chi(v_{i,a_i}) + (c-1)2^{d-i}$ and $\chi(v_{i',b_{i'}}) = \chi(v_{i',a_{i'}}) + (c-1)2^{d-i'}$,

$$\begin{aligned}|\chi(v_{i,b_i}) - \chi(v_{i',b_{i'}})| &> (c-1)2^{d-i} - (c-1)2^{d-i'} - 2^{d-i} \\ &\geq (c-1)2^{d-i} - \frac{c-1}{2} \cdot 2^{d-i} - 2^{d-i} = (c/2 - 3/2)2^{d-i} \geq 2^{d-i},\end{aligned}$$

if $c \geq 5$. Therefore, v_{i,b_i} and $v_{i',b_{i'}}$ are independent (Prop. 5.4). Similarly, if a large edge E_k covers both $v_{b_j,j}$ and $v_{b_{j'},j'}$ then $v_{a_j,j}$ and $v_{a_{j'},j'}$ are independent.

If X can be covered with weight less than $\sqrt{d}/2$, then there is an edge that covers at least $|X|/(\sqrt{d}/2) = 2\sqrt{d}$ vertices of X. Denote by $Y \subseteq X$ this set of vertices, and let $Y_a = \{v_{i,a_i} \in Y : d/2 \leq i \leq d\}$ and $Y_b = \{v_{i,b_i} \in Y : d/2 \leq i \leq d\}$. Now either $|Y_a| \geq \sqrt{d}$ or $|Y_b| \geq \sqrt{d}$. For each vertex v_{i,a_i}, we call the vertex v_{i,b_i} the *pair* of v_{i,a_i} and vice versa. If $|Y_a| \geq \sqrt{d}$, then we have seen that the pairs of the vertices in Y_a form an independent set of size $|Y_a|$, thus X cannot be covered with weight less than $\sqrt{d}$. Similarly, if $|Y_b| \geq \sqrt{d}$, then the pairs of the vertices in Y_b give an independent set of size $\sqrt{d}$. This contradicts the assumption that S can be covered with weight strictly less than $\sqrt{d}/2$.

Thus there is a $d/2 \leq i \leq d$ such that for every $0 \leq j \leq 2^i - c$, at least one of $v_{i,j}, \ldots, v_{i,j+c-1}$ is not in S. It is not difficult to see that the set C_i of vertices on level i not in S is connected and intersects more than $1/(2c)$ fraction of the large edges. Thus $\text{weight}(\gamma|C) > \text{weight}(\gamma)/2c$ for the component C of $H(d,c) \setminus S$ containing C_i, contradicting the assumption that S is a $\frac{1}{2c}$-balanced separator for γ. ∎

Corollary 5.8. *$\mathcal{H}_c$ has unbounded fractional hypertree width for every $c \geq 5$.*

5.2. Upper bound on adaptive width

We use the following lemma to give an upper bound for f-width (proof is omitted):

Lemma 5.9. *Let H be a hypergraph, $0 < \lambda < 1, w > 0$ constants, and $f : 2^{V(H)} \to \mathbb{R}^+$ a function such that $f(X) \leq f(Y)$ for every $X \subseteq Y$ and $f(X \cup Y) \leq f(X) + f(Y)$ for arbitrary X, Y. Assume that for every subset $W \subseteq V(H)$ there is a subset $S \subseteq V(H)$ with $f(S) \leq w$ such that every component C of $H \setminus S$ has $f(C \cap W) \leq \lambda f(W)$. Then the f-width of H is at most $2w/(1-\lambda) + w$.*

To obtain the upper bound on adaptive width, we have to show that the required separator S exists for every fractional independent set. We say that a set S is *closed* if the set S contains every ancestor of every vertex of S. For future use, we show that even a closed separator exists for $H(d,c)$.

Lemma 5.10. *Let ϕ be a fractional independent set of $H(d,c)$ and let W be a subset of vertices. Then there is a closed set S with $\phi(S) \leq 4c(c+1)+5$ such that for every component C of $H(d,c) \setminus S$ we have $\phi(C \cap W) \leq 3\phi(W)/4$.*

Proof. Let $M(a,b)$ be the set of vertices $v_{i,j}$ with $a \leq \chi(v_{i,j}) < b$. Let x and y be integers such that $\phi(M(x, x+y) \cap W) \geq \phi(W)/4$ and y is as small as possible. Let $d_0 = d - \lceil \log_2 y \rceil$; clearly, we have $y \leq 2^{d-d_0} \leq 2y$. Let $A(t) := \{v_{i,j} : \chi(v_{i,j}) \geq t \text{ and } i \geq d_0\}$. Denote by $S(t)$ the set of those vertices v that have a descendant $v_{i,j}$ with $\chi(v_{i,j}) < t$ such that $v_{i,j}$ has a neighbor in $A(t)$. We show that $\phi(S(t_1)) \leq 2c(c+1)+1$ for some $x - y < t_1 \leq y$.

Let $S_1(t)$ be those vertices of $S(t)$ that are on level less than d_0 and let $S_2(t)$ be those vertices that are on level at least d_0. First we bound $\phi(S_1(t))$. Observe that every $v \in S_1(t)$ has a descendant $v_{i,j}$ with $\chi(v_{i,j}) \geq t - c2^{d-d_0}$: if descendant $v_{i,j}$ has a neighbor $u \in A(t)$, then either $v_{i,j}$ and u are connected by a large edge (in this case u is also a descendant of v) or $v_{i,j}$ and u are connected by a small edge (in this case $\chi(v_{i,j}) \geq t - c2^{d-i} \geq t - c2^{d-d_0}$). Let X be the set of vertices $v_{d_0,j}$ with $t - c2^{d_0} \leq \chi(v_{d_0,j}) < t$, we have $|X| \leq c$. By the observation above, every $v \in S_1(t)$ has a descendant in X. The vertices in X and the ancestors of X can be covered by $|X| \leq c$ large edges. Thus $\phi(S_1(t)) \leq c$, as $S_1(t)$ can be covered with at most c large edges and $\phi(E_k) \leq 1$ for every large edge E_k.

We show that $\phi(S_2(t))$ is small on average. We claim that

$$\sum_{t=x-y+1}^{x} \phi(S_2(t)) \leq c \sum_{t=\max\{0,(x-y-c2^{d-d_0})\}}^{\min\{2^d, x+2^{d-d_0}-1\}} \phi(E_t) \leq c(c+1)2^{d-d_0} + y \leq 2c(c+1)y + y.$$

holds, implying that $\phi(S_2(t)) \leq 2c(c+1)+1$ for at least one t. To see the first inequality, observe that $v_{i,j}$ with $i \geq d_0$ is in $S_2(t)$ only if $t - c2^{d-i} \leq \chi(v_{i,j}) < t$. Thus such a vertex contributes to the first sum for at most $c2^{d-i}$ values of t. However, if $v_{i,j}$ contributes at all

to the first sum, then it contributes to the second sum for exactly 2^{d-i} values of t, as every large edge containing $v_{i,j}$ is counted. Thus $v_{i,j}$ contributes $\phi(v_{i,j})$ at most c times more to the first sum than to the second, which is taken care by the factor c before the second sum.

Similarly, we can show that there is a value $x+y \leq t_2 < x+2y$ such that $\phi(S_2(t_2)) \leq 2c(c+1)+1$. Denote by $T(t_1,t_2)$ the vertices of $M(t_1,t_2)$ on level less than d_0. We claim that $\phi(T(t_1,t_2)) \leq 3$. First, $T(t_1,t_2)$ can contain at most 3 vertices on each level: if $v_{i,j}, v_{i,j'} \in T(t_1,t_2)$ and $j' \geq j+3$, then $|\chi(v_{i,j})-\chi(v_{i,j'})| \geq 3\cdot 2^{d-i} > 3\cdot 2^{d-d_0} \geq 3y \geq t_2-t_1$, contradicting the assumption on the χ-values. Every $v_{i,j} \in T(t_1,t_2)$ has a descendant $v_{i',j'} \in T(t_1,t_2)$ for every $i < i' < d_0$, namely $v_{i',j'}$ with $j' = j2^{i'-i}$. Thus by covering the at most 3 vertices of $T(t_1,t_2)$ on level d_0-1 by at most 3 large edges, we can cover $T(t_1,t_2)$, and $\phi(T(t_1,t_2)) \leq 3$ follows.

Define $S := S(t_1) \cup S(t_2) \cup T(t_1,t_2)$. Clearly, $\phi(S) \leq 2(2c(c+1)+1)+3 = 4c(c+1)+5$. We show that S separates $M(t_1,t_2)$ from the rest of the vertices. Suppose that $v_{i,j}, v_{i',j'} \notin S$ are adjacent vertices such that $v_{i,j} \in M(t_1,t_2)$ and $v_{i',j'} \notin M(t_1,t_2)$. We have $i \geq d_0$ (otherwise $v_{i,j} \in T(t_1,t_2)$), hence $v_{i,j} \in A(t_1)$. If $\chi(v_{i',j'}) < t_1$, then $v_{i',j'} \in S(t_1) \subseteq S$, a contradiction. Moreover, if $\chi(v_{i',j'}) > t_2$, then $i' \geq d_0$ as $v_{i,j}$ and $v_{i',j'}$ are not neighbors if $\chi(v_{i,j}) < \chi(v_{i',j'})$ and $i > i'$. Thus $v_{i',j'} \in A(t_2)$ and $v_{i,j} \in S(t_2) \subseteq S$, a contradiction.

By the definition of x and y, we have $\phi(M(t_1,t_2) \cap W) \geq \phi(M(x,x+y) \cap W) \geq \phi(W)/4$. To complete the proof that $\phi(W \cap C) \leq 3\phi(W)/4$ for every component C of $H(d,c) \setminus S$, we show that $\phi(M(t_1,t_2) \cap W) \leq 3\phi(W)/4$: as we have seen that every such component C is fully contained in either $M(t_1,t_2)$ or $V \setminus M(t_1,t_2)$, this means that no component C can have $\phi(W \cap C) > 3\phi(W)/4$. Since $x-t_1 < y$, the minimality of y implies $\phi(M(t_1,x) \cap W) \leq \phi(W)/4$. Similarly, it follows from $t_2-(x+y) < y$ that $\phi(M(x+y,t_2) \cap W) \leq \phi(W)/4$. Now $\phi(M(t_1,t_2) \cap W) = \phi(M(t_1,x) \cap W) + \phi(M(x,x+y) \cap W) + \phi(M(x+y,t_2) \cap W) \leq \frac{3}{4}\phi(W)$. ■

By Lemma 5.10, the requirements of Lemma 5.9 hold for $H(d,c)$ with $w := 4c(c+1)+5$ and $\lambda := 3/4$, hence $\operatorname{adw}(H(d,c)) \leq 9w = 36c(c+1)+45$.

Corollary 5.11. *The class $\mathcal{H}_c$ has bounded adaptive width for every fixed $c \geq 1$.*

References

[1] I. Adler. *Width functions for hypertree decompositions*. PhD thesis, 2006.

[2] H. Chen and M. Grohe. Constraint satisfaction problems with succinctly specified relations, 2006.

[3] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[4] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, Berlin, 2006.

[5] E. C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *Proc. of AAAI-90*, 4–9, Boston, MA, 1990.

[6] G. Gottlob, F. Scarcello, and M. Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artificial Intelligence*, 138(1-2):55–86, 2002.

[7] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1, 2007.

[8] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA '06*, 289–298, 2006.

[9] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

[10] D. Marx. Approximating fractional hypertree width. In *SODA '09*, 902–911, 2009.

[11] D. Marx. Can you beat treewidth? In *FOCS '07*, 169–179, 2007.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 661–672
www.stacs-conf.org

BÜCHI COMPLEMENTATION MADE TIGHT

SVEN SCHEWE

University of Liverpool
E-mail address: Sven.Schewe@liverpool.ac.uk
URL: http://www.csc.liv.ac.uk/research/logics/

Abstract. The precise complexity of complementing Büchi automata is an intriguing and long standing problem. While optimal complementation techniques for finite automata are simple – it suffices to determinize them using a simple subset construction and to dualize the acceptance condition of the resulting automaton – Büchi complementation is more involved. Indeed, the construction of an EXPTIME complementation procedure took a quarter of a century from the introduction of Büchi automata in the early 60s, and stepwise narrowing the gap between the upper and lower bound to a simple exponent (of $(6e)^n$ for Büchi automata with n states) took four decades. While the distance between the known upper $(O((0.96\,n)^n))$ and lower $(\Omega((0.76\,n)^n))$ bound on the required number of states has meanwhile been significantly reduced, an exponential factor remains between them. Also, the upper bound on the size of the complement automaton is not linear in the bound of its state space. These gaps are unsatisfactory from a theoretical point of view, but also because Büchi complementation is a useful tool in formal verification, in particular for the language containment problem. This paper proposes a Büchi complementation algorithm whose complexity meets, modulo a quadratic $(O(n^2))$ factor, the known lower bound for Büchi complementation. It thus improves over previous constructions by an exponential factor and concludes the quest for optimal Büchi complementation algorithms.

1. Introduction

The precise complexity of Büchi complementation is an intriguing problem for two reasons: First, the quest for optimal algorithms is a much researched problem (c.f., [Büc62, SS78, Péc86, SVW87, Saf88, Mic88, Tho99, Löd99, KV01, GKSV03, FKV06, Pit07, Var07, Yan08]) that has defied numerous approaches to solving it. And second, Büchi complementation is a valuable tool in formal verification (c.f., [Kur94]), in particular when studying language inclusion problems of ω-regular languages. In addition to this, complementation is useful to check the correctness of other translation techniques [Var07, TCT+08]. The GOAL tool [TCT+08], for example, provides such a test suite and incorporates four of the more recent algorithms [Saf88, Tho99, KV01, Pit07] for Büchi complementation.

1998 ACM Subject Classification: F.4 Mathematical Logic and Formal Languages.

Key words and phrases: Automata and Formal Languages, Büchi Complementation, Automata Theory, Nondeterministic Büchi Automata.

This work was partly supported by the EPSRC through the grand EP/F033567/1 *Verifying Interoperability Requirements in Pervasive Systems*.

While devising optimal complementation algorithms for nondeterministic finite automata is simple—nondeterministic *finite* automata can be determinized using a simple subset construction, and deterministic finite automata can be complemented by complementing the set of final states [RS59, SS78]—devising optimal complementation algorithms for nondeterministic Büchi automata is hard, because simple subset constructions are not sufficient to determinize or complement them [Mic88, Löd99].

Given the hardness and importance of the problem, Büchi complementation enjoyed much attention [Büc62, Péc86, SVW87, Mic88, Saf88, Löd99, Tho99, KV01, GKSV03, FKV06, Var07, TCT+08, Yan08], resulting in a continuous improvement of the upper and lower bounds.

The first complementation algorithm dates back to the introduction of Büchi automata in 1962. In his seminal paper "On a decision method in restricted second order arithmetic" [Büc62], Büchi develops a complementation procedure that comprises a doubly exponential blow-up. While Büchi's result shows that nondeterministic Büchi automata (and thus ω-regular expressions) are closed under complementation, complementing an automaton with n states may, when using Büchi's complementation procedure, result in an automaton with $2^{2^{O(n)}}$ states, while an $\Omega(2^n)$ lower bound [SS78] is inherited from finite automata.

In the late 80s, these bounds have been improved in a first sequence of results, starting with establishing an EXPTIME upper bound [Péc86, SVW87], which matches the EXPTIME lower bound [SS78] inherited from finite automata. However, the early EXPTIME complementation techniques produce automata with up to $2^{O(n^2)}$ states [Péc86, SVW87]; hence, these upper bounds are still exponential in the lower bounds.

This situation changed in 1988, when Safra introduced his famous determinization procedure for nondeterministic Büchi automata [Saf88], resulting in an $n^{O(n)}$ bound for Büchi complementation, while Michel [Mic88] established a seemingly matching $\Omega(n!)$ lower bound in the same year. Together, these results imply that Büchi complementation is in $n^{\theta(n)}$, leaving again the impression of a tight bound.

As pointed our by Vardi [Var07], this impression is misleading, because the $O()$ notation hides an $n^{\theta(n)}$ gap between both bounds. This gap has been narrowed down in 2001 to $2^{\theta(n)}$ by the introduction of an alternative complementation technique that builds on level rankings and a cut-point construction [KV01]. (Level rankings are functions from the states Q of a nondeterministc Büchi automaton to $\{0, 1, \ldots, 2|Q|+1\}$.) The complexity of the plain method is approximately $(6n)^n$ [KV01], leaving a $(6e)^n$ gap to Michel's lower bound [Mic88].

Recently, tight level rankings [FKV06, Yan08]—a special class of level rankings that is onto a predefined subset—have been exploited by Friedgut, Kupferman, and Vardi [FKV06] to improved the upper complexity bound to $O\big((0.96n)^n\big)$, and by Yan [Yan08] to improve the lower complexity bound to $\Omega\big((0.76n)^n\big)$.

In the remainder of this paper, we first recapitulate the basic complementation technique of Kupferman and Vardi [KV01], and discuss the core ideas of the improved complexity analysis of Friedgut, Kupferman, and Vardi [FKV06] and Yan [Yan08]. We then show how to improve the complementation technique of Friedgut, Kupferman, and Vardi [FKV06] such that the resulting complementation algorithm meets the known lower bound [Yan08] modulo a small polynomial factor (quadratic in the size of the automaton that is to be complemented), and show that, different to older constructions [KV01, GKSV03], we can achieve an equivalent bound on the number of edges.

2. Preliminaries

2.1. Büchi Automata

Nondeterministic Büchi automata [Büc62] are used to represent ω-regular languages $L \subseteq \Sigma^\omega = \omega \to \Sigma$ over a finite alphabet Σ. A nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ is a five tuple, consisting of a finite alphabet Σ, a finite set Q of states with a non-empty subset $I \subseteq Q$ of initial states, a transition function $\delta : Q \times \Sigma \to 2^Q$ that maps states and input letters to sets of successor states, and a set $F \subseteq Q$ of final states.

Nondeterministic Büchi automata are interpreted over infinite sequences $\alpha : \omega \to \Sigma$ of input letters. An infinite sequence $\rho : \omega \to Q$ of states of $\mathcal{A}$ is called a *run* of $\mathcal{A}$ on an input word α if the first letter $\rho(0) \in I$ of ρ is an initial state, and if, for all $i \in \omega$, $\rho(i+1) \in \delta\big(\rho(i), \alpha(i)\big)$ is a successor state of $\rho(i)$ for the input letter $\alpha(i)$.

A run $\rho : \omega \to Q$ is called *accepting* if some finite state appears infinitely often in ρ ($inf(\rho) \cap F \neq \emptyset$ for $inf(\rho) = \{q \in Q \mid \forall i \in \omega\ \exists j > i$ such that $\rho(j) = q\}$). A word $\alpha : \omega \to \Sigma$ is *accepted* by $\mathcal{A}$ if $\mathcal{A}$ has an accepting run on α, and the set $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^\omega \mid \alpha$ is accepted by $\mathcal{A}\}$ of words accepted by $\mathcal{A}$ is called its *language*.

For technical convenience we also allow for finite runs $q_0 q_1 q_2 \dots q_n$ with $\delta\big(q_n, \alpha(n)\big) = \emptyset$. Naturally, no finite run satisfies the Büchi condition; all finite runs are therefore rejecting, and have no influence on the language of an automaton.

The two natural complexity measures for a Büchi automaton are the size $|Q|$ of its state space, and its size $\sum_{q \in Q,\ \sigma \in \Sigma} 1 + |\delta(q, \sigma)|$, measured in the size of its transition function.

2.2. Run DAG and Acceptance

In [KV01], Kupferman and Vardi introduce a Büchi complementation algorithm that uses level rankings as witnesses for the absence of an accepting run.

The set of all runs of a nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ on a word $\alpha : \omega \to \Sigma$ can be represented by a directed acyclic graph (DAG) $\mathcal{G}_\alpha = (V, E)$ with

- vertices $V \subseteq Q \times \omega$ such that $(q, p) \in V$ is in the set V of vertices if and only if there is a run ρ of $\mathcal{A}$ on α with $\rho(p) = q$, and
- edges $E \subseteq (Q \times \omega) \times (Q \times \omega)$ such that $\big((q, p), (q', p')\big) \in E$ if and only if $p' = p + 1$ and $q' \in \delta\big(q, \alpha(p)\big)$ is a successor of q for the input letter $\alpha(p)$.

We call $\mathcal{G}_\alpha = (V, E)$ the *run DAG* of $\mathcal{A}$ for α, and the vertices $V \cap (Q \times \{i\})$ of $\mathcal{G}_\alpha = (V, E)$ that refer to the i^{th} position of runs the i^{th} *level* of $\mathcal{G}_\alpha = (V, E)$.

The run DAG $\mathcal{G}_\alpha$ is called *rejecting* if no path in $\mathcal{G}_\alpha$ satisfies the Büchi condition. That is, $\mathcal{G}_\alpha$ is rejecting if and only if $\mathcal{A}$ rejects α. $\mathcal{A}$ can therefore be complemented to a nondeterministc Büchi automaton $\mathcal{B}$ that checks if $\mathcal{G}_\alpha$ is rejecting.

The property that $\mathcal{G}_\alpha$ is rejecting can be expressed in terms of ranks. We call a vertex $(q, p) \in V$ of a DAG $\mathcal{G} = (V, E)$ *finite*, if the set of vertices reachable from (q, p) in $\mathcal{G}$ is finite, and *endangered*, if no vertex reachable from (q, p) is accepting (that is, in $F \times \omega$).

Based on these definitions, *ranks* can be assigned to the vertices of a rejecting run DAG. We set ${\mathcal{G}_\alpha}^0 = \mathcal{G}_\alpha$, and repeat the following procedure until a fixed point is reached, starting with $i = 0$:

- Assign all finite vertices of ${\mathcal{G}_\alpha}^i$ the rank i, and set ${\mathcal{G}_\alpha}^{i+1}$ to ${\mathcal{G}_\alpha}^i$ minus the states with rank i (that is, minus the states finite in ${\mathcal{G}_\alpha}^i$).

- Assign all endangered vertices of ${\mathcal{G}_\alpha}^{i+1}$ the rank $i+1$, and set ${\mathcal{G}_\alpha}^{i+2}$ to ${\mathcal{G}_\alpha}^{i+1}$ minus the states with rank $i+1$ (that is, minus the states endangered in ${\mathcal{G}_\alpha}^{i+1}$).
- Increase i by 2.

A fixed point is reached in $n+1$ steps, and the ranks can be used to characterize the complement language of a nondeterministic Büchi automaton:

Proposition 2.1. [KV01] *A nondeterministic Büchi automaton $\mathcal{A}$ with n states rejects a word $\alpha : \omega \to \Sigma$ if and only if ${\mathcal{G}_\alpha}^{2n+1}$ is empty.* ∎

To see that a fixed point is reached after $n+1$ iterations, note that deleting all finite or endangered vertices leaves a DAG without finite or endangered vertices, respectively. Hence a fixed point is reached as soon as we do not assign a rank i to any vertex. By construction, no DAG ${\mathcal{G}_\alpha}^{2i+1}$ contains finite vertices. If it contains an endangered vertex v, then all vertices reachable from v are endangered, too. This implies that some vertex of almost all levels is assigned the rank $2i+1$. Hence, if no fixed point is reached earlier, some rank is assigned to all vertices of almost all levels after n iterations (there cannot be more than n vertices in a level), and all vertices in ${\mathcal{G}_\alpha}^{2n}$ must be finite.

If the reached fixed point ${\mathcal{G}_\alpha}^{\infty}$ is non-empty, then it contains only infinite and non-endangered vertices, and constructing an accepting run from ${\mathcal{G}_\alpha}^{\infty}$ is simple. Vice versa, an accepting run ρ on an ω-word α can be viewed as an infinite sub-graph of $\mathcal{G}_\alpha$ that does not contain finite or endangered nodes. By a simple inductive argument, the sub-graph identified by ρ is therefore a sub-graph of ${\mathcal{G}_\alpha}^{i}$ for all $i \in \omega$, and hence of ${\mathcal{G}_\alpha}^{\infty}$.

2.3. Büchi Complementation

The connection between Büchi complementation, run DAGs and ranks leads to an elegant complementation technique. We call the maximal rank of a vertex in a level the rank of this level, the rank of almost all vertices $(\rho(i), i)$, $i \in \omega$ of a run ρ (or: path in $\mathcal{G}$) the rank of ρ, and the rank of almost all levels of a DAG $\mathcal{G}$ the rank of $\mathcal{G}$. (Note that level ranks can only go down, and that vertex ranks can only go down along a path.)

For a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ with n states, we call a function $f : Q \to \{0, 1, \ldots, 2n\}$ that maps all accepting states to odd numbers $(f(F) \cap 2\omega = \emptyset)$ a *level ranking*.

Proposition 2.2. [KV01] *For a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, the nondeterministic Büchi automaton $\mathcal{B} = (\Sigma, Q', I', \delta', F')$ with*

- $Q' = 2^Q \times 2^Q \times \mathcal{R}$,
- $I' = \{I\} \times \{\emptyset\} \times \mathcal{R}$,
- $\delta'\big((S, O, f), \sigma\big) = \big\{\big(\delta(S, \sigma), \delta(O, \sigma) \smallsetminus \mathit{odd}(f'), f'\big) \mid f' \leq^S_\sigma f,\ O \neq \emptyset\big\}$
  $\cup \big\{\big(\delta(S, \sigma), \delta(S, \sigma) \smallsetminus \mathit{odd}(f'), f'\big) \mid f' \leq^S_\sigma f,\ O = \emptyset\big\}$, *and*
- $F' = 2^Q \times \{\emptyset\} \times \mathcal{R}$,

where

- $\mathcal{R}$ *is the set of all level rankings of* $\mathcal{A}$,
- $\mathit{odd}(f) = \{q \in Q \mid f(q)$ *is odd*$\}$, *and*
- $f' \leq^S_\sigma f :\Leftrightarrow \forall q \in S,\ q' \in \delta(q, \sigma).\ f'(q') \leq f(q)$,

accepts the complement $\mathcal{L}(\mathcal{B}) = \overline{\mathcal{L}(\mathcal{A})} = \Sigma^\omega \smallsetminus \mathcal{L}(\mathcal{A})$ of the language of $\mathcal{A}$. ∎

The first element S of a triple $(S, O, f) \in Q'$ reflects the set of states of $\mathcal{A}$ reachable upon the input seen so far (the states in the respective level of the run DAG), and the third element f is a mapping that intuitively maps reachable states to their rank.

The condition $f' \leq_\sigma^S f$ ensures that the rank of the vertices (or rather: the value assigned to them) is decreasing along every path of the run DAG. The second element is used for a standard cut-point construction, comparable to the cut-point constructions in the determinization of Co-Büchi automata or the nondeterminization of alternating Büchi automata. It contains the positions whose rank (or rather: the value assigned to them) has been even ever since the last cut-point ($O = \emptyset$) was reached; it intuitively ensures that the respective vertices are finite.

2.4. Tight Level Rankings

Friedgut, Kupferman, and Vardi [FKV06] improved this complementation technique by exploiting the observation that the true ranks of the run DAG $\mathcal{G}_\alpha$ of a rejected ω-word α are eventually always tight. A level ranking $f : Q \to \omega$ is called *tight*, if it has an odd rank r, and is onto the odd numbers $\{1, 3, \ldots, r\}$ up to its rank r, and S-*tight*, if its restriction to S is tight and if it maps all states not in S to 1 ($f(q) = 1 \ \forall q \in Q \setminus S$).

Proposition 2.3. [FKV06] *For every run DAG $\mathcal{G}_\alpha$ with finite rank r, it holds that*

- *r is odd, and*
- *there is a level $l \geq 0$ such that, for all levels $l' \geq l$ and all odd ranks $o \leq r$, there is a node $(q, l') \in \mathcal{G}_\alpha$ with rank o in $\mathcal{G}_\alpha$.* ∎

This immediately follows from what was said in Subsection 2.2 on reaching a fixed point after $n+1$ iterations: If the rank $\mathcal{G}_\alpha$ is r then, for every odd number $o \leq r$, almost all levels contain a vertex with rank o, and assuming that r is even implies that all vertices of ${\mathcal{G}_\alpha}^r$ are finite, which in turn implies that only finitely many levels contain a vertex with rank r and hence leads to a contradiction.

Using this observation, the construction from Proposition 2.2 can be improved by essentially replacing $\mathcal{R}$ by the set $\mathcal{T} = \{f \in \mathcal{R} \mid f \text{ is tight}\}$ of tight level rankings. While the size $|\mathcal{R}| = (2n+1)^n$ of $\mathcal{R}$ is in $\theta\big((2n)^n\big)$, the size of $\mathcal{T}$,

$$\mathit{tight}(n) = |\mathcal{T}|,$$

is much smaller. Building on an approximation of Stirling numbers of the second kind by Temme [Tem93], Yan and Friedgut, Kupferman, and Vardi [FKV06, Yan08] showed that $\mathit{tight}(n)$ can be approximated by $(\kappa n)^n$ for a constant $\kappa \approx 0.76$, that is, they showed

$$\kappa = \lim_{n\to\infty} \frac{\sqrt[n]{\mathit{tight}(n)}}{n} \approx 0.76.$$

Friedgut, Kupferman, and Vardi [FKV06] use this observation—together with other improvements—for an improved complementation algorithm that produces a complement automaton with approximately $(0.96\,n)^n$ states [FKV06].

Yan [Yan08] showed for full automata—a family of automata that has exactly one accepting state, and an alphabet that encodes the possible transitions between the states of the automaton—that every nondeterministic Büchi automaton that accepts the complement language of a full automaton with $n+1$ states must have $\Omega\big(\mathit{tight}(n)\big)$ states.

Proposition 2.4. [Yan08] *A nondeterministic Büchi automaton that accepts the complement language of a full Büchi automaton with n states has $\Omega\big(\mathit{tight}(n-1)\big)$ states.* ∎

3. Efficient Büchi Complementation

To optimize the construction from Proposition 2.2, we turn not only to tight functions (c.f. [FKV06]), but also refine the cut-point construction. While the cut-point construction of Proposition 2.2 tests *concurrently* for all even ranks if a path has finite even rank, we argue that it is much cheaper to test this property *turn wise* for all even ranks individually. As a result, the overall construction becomes more efficient and meets, modulo a small polynomial factor in $O(n^2)$, the lower bound recently established by Yan [Yan08].

3.1. Construction

The obtained state space reduction of the proposed construction compared to [FKV06] is due to an efficient cut-point construction in combination with the restriction to tight rankings. The improved cut-point construction is inspired by the efficient translation from generalized to ordinary Büchi automata. Indeed, the acceptance condition that no trace has an *arbitrary* even rank, which is reflected by the straight-forward acceptance condition of previous algorithms [KV01, FKV06], can be replaced by an acceptance condition, which only rules out that some trace has a particular even rank, but does so for all potential even ranks.

Checking the condition for a particular even rank allows for focusing on exactly this rank in the cut-point construction, which led to a significant cut in the size of the resulting state space. While this approach cannot be taken if we literally use a generalized Büchi condition, the idea of cyclically considering the relevant even ranks proves to be feasible.

Construction: For a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ with $n = |Q|$ states, let $\mathcal{C} = (\Sigma, Q', I', \delta', F')$ denote the nondeterministic Büchi automaton with

- $Q' = Q_1 \cup Q_2$ with
  - $Q_1 = 2^Q$ and
  - $Q_2 = \{(S, O, f, i) \in 2^Q \times 2^Q \times \mathcal{T} \times \{0, 2, \ldots, 2n-2\} \mid$ f is S-tight, $O \subseteq S$ and $\exists i \in \omega.\ O \subseteq f^{-1}(2i)\}$,
- $I' = \{I\}$,
- $\delta' = \delta_1 \cup \delta_2 \cup \delta_3$ for
  - $\delta_1 : Q_1 \times \Sigma \to 2^{Q_1}$ with $\delta_1(S, \sigma) = \{\delta(S, \sigma)\}$,
  - $\delta_2 : Q_1 \times \Sigma \to 2^{Q_2}$ with $(S', O, f, i) \in \delta_2(S, \sigma) \Leftrightarrow S'{=}\delta(S, \sigma)$, $O{=}\emptyset$, and $i{=}0$,
  - $\delta_3 : Q_2 \times \Sigma \to 2^{Q_2}$ with $(S', O', f', i') \in \delta_3\big((S, O, f, i), \sigma\big)$ $\Leftrightarrow$ $S' = \delta(S, \sigma)$, $f' \leq^S_\sigma f$, $\mathit{rank}(f) = \mathit{rank}(f')$, and
    - $i' = (i + 2) \mod (\mathit{rank}(f') + 1)$ and $O' = {f'}^{-1}(i')$ if $O = \emptyset$ or
    - $i' = i$ and $O' = \delta(O, \sigma) \cap {f'}^{-1}(i)$ if $O \neq \emptyset$, respectively, and
- $F' = \{\emptyset\} \cup (2^Q \times \{\emptyset\} \times \mathcal{T} \times \omega) \cap Q_2$.

The complement automaton $\mathcal{C}$ operates in two phases. In a first phase it only traces the states reachable in $\mathcal{A}$ upon a finite input sequence. In this phase, only the states in Q_1 and the transition function δ_1 are used. In the special case that $\mathcal{A}$ rejects an ω-word α because $\mathcal{A}$ has no run on α, $\mathcal{C}$ accepts by staying forever in phase one, because $\{\emptyset\}$ is final.

$\mathcal{C}$ intuitively uses its nondeterministic power to guess a point in time where all successive levels are tight and have the same rank. At such a point, $\mathcal{C}$ traverses from Q_1 to Q_2, using a transition from δ_2. Staying henceforth in Q_2 (using the transitions from δ_3), $\mathcal{C}$ intuitively verifies turn wise for all potential even ranks e that no path has this particular even rank e. For a particular rank e, it suffices to trace the positions on traces with unchanged rank e (hence $O \subseteq f^{-1}(e)$), and to cyclically update the designated even rank after every cut-point.

3.2. Correctness

To show that the automaton $\mathcal{C}$ from the construction introduced in the previous subsection recognises the complement language of $\mathcal{A}$, we first show that the complement language of $\mathcal{C}$ contains the language of $\mathcal{A}$ ($\mathcal{L}(\mathcal{A}) \subseteq \overline{\mathcal{L}(\mathcal{C})}$), and then that the complement language of $\mathcal{A}$ is contained in the language of $\mathcal{C}$ ($\overline{\mathcal{L}(\mathcal{A})} \subseteq \mathcal{L}(\mathcal{C})$).

Lemma 3.1. *If a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ accepts an ω-word $\alpha : \omega \to \Sigma$, then α is rejected by the automaton $\mathcal{C} = (\Sigma, Q', I', \delta', F')$. ($\mathcal{L}(\mathcal{A}) \subseteq \overline{\mathcal{L}(\mathcal{C})}$)*

Proof. Let $\rho : \omega \to Q$ be an accepting run of $\mathcal{A}$ on α. First, $\rho' = S_0, S_1, S_2 \ldots$ with $S_0 = I$ and $S_{i+1} = \delta\big(S_i, \alpha(i)\big)$ for all $i \in \omega$ is no accepting run of $\mathcal{C}$, because $\rho(i) \in S_i$ and hence no state of ρ' is accepting ($\emptyset \neq S_i \notin F'$ for all $i \in \omega$). Let now

$$\rho' = S_0, S_1, S_2, \ldots, S_p, (S_{p+1}, O_{p+1}, f_{p+1}, i_{p+1}), (S_{p+2}, O_{p+2}, f_{p+2}, i_{p+2}), \ldots$$

be a run of $\mathcal{C}$ on α. Again, we have that $\rho(j) \in S_j$ for all $j \in \omega$. Furthermore, the construction guarantees that $f_{j+1} \leq^S_{\alpha(j)} f_j$ holds for all $j > p$. The sequence

$$f_{p+1}(\rho(p+1)\big) \geq f_{p+2}(\rho(p+2)\big) \geq f_{p+3}(\rho(p+3)\big) \geq \ldots$$

is therefore decreasing, and stabilizes eventually. That is, there is a $k > p$ and a $v \leq 2n$ such that $f_l\big(\rho(l)\big) = v$ for all $l \geq k$. Since ρ is accepting, there is a position $l \geq k$ with $\rho(l) \in F$. Taking into account that f_l is a level ranking, this implies that $f_l\big(\rho(l)\big)$—and hence v—is even. Assuming that ρ' is accepting, we can infer that, for some position $l > k$ which follows one of the first n accepting states of ρ' after position k, $i_l = v$ and $O_l = f_l^{-1}(v) \ni \rho(l)$. It is now easy to show by induction that, for all $m \geq l$, $i_m = v$ and (using $f_m\big(\rho(m)\big) = v$)) $\rho(m) \in O_m \neq \emptyset$ hold true, which contradicts the assumption that ρ' is accepting. ∎

To proof the second lemma, $\overline{\mathcal{L}(\mathcal{A})} \subseteq \mathcal{L}(\mathcal{C})$, we use Propositions 2.1 and 2.3 to infer that the run DAG $\mathcal{G}_\alpha$ of an ω-word rejected by $\mathcal{A}$ is either finite or has odd bounded rank and only finitely many non-tight level rankings. We use this to build an accepting run of $\mathcal{C}$ on α.

Lemma 3.2. *For a nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, the automaton $\mathcal{C} = (\Sigma, Q', I', \delta', F')$ accepts an ω-word $\alpha : \omega \to \Sigma$ if α is rejected by $\mathcal{A}$. ($\overline{\mathcal{L}(\mathcal{A})} \subseteq \mathcal{L}(\mathcal{C})$)*

Proof. If $\alpha : \omega \to \Sigma$ is rejected by $\mathcal{A}$, then the run DAG $\mathcal{G}_\alpha$ has bounded rank by Proposition 2.1, and by Proposition 2.3 almost all levels of $\mathcal{G}_\alpha$ have tight level rankings with the same rank r. For the special case that the rank of all vertices of $\mathcal{G}_\alpha$ is 0, that is, if all vertices of $\mathcal{G}_\alpha$ are finite, $\rho' = S_0, S_1, S_2 \ldots$ with $S_0 = I$ and $S_{i+1} = \delta\big(S_i, \alpha(i)\big)$ for all $i \in \omega$ is an accepting run of $\mathcal{C}$ on α.

If $\mathcal{G}_\alpha$ contains an infinite vertex, then we fix a position $p \in \omega$ such that the rank of all levels $p' \geq p$ of $\mathcal{G}_\alpha$ is r and tight for some (odd) $r \geq 1$. We now consider a run

$\rho' = S_0, S_1, S_2, \ldots, S_p, (S_{p+1}, O_{p+1}, f_{p+1}, i_{p+1}), (S_{p+2}, O_{p+2}, f_{p+2}, i_{p+2}), \ldots$ of $\mathcal{C}$ on α with

- $S_0 = I$, $O_{p+1} = \emptyset$, and $i_{p+1} = 0$,
- $S_{j+1} = \delta\big(S_j, \alpha(j)\big)$ for all $j \in \omega$, and
- $O_{j+1} = f_{j+1}^{-1}(i_{j+1})$ if $O_j = \emptyset$ or
  $O_{j+1} = \delta\big(O_j, \alpha(j)\big) \cap f_{j+1}^{-1}(i_{j+1})$ if $O_j \neq \emptyset$, respectively, for all $j > p$,
- f_j is the S_j-tight level ranking that maps each state $q \in S_j$ to the rank of (q, j) $\forall j > p$,
- $i_{j+1} = i_j$ if $O_j \neq \emptyset$ or
  $i_{j+1} = (i_j + 2) \mod (\mathit{rank}(f) + 1)$ if $O_j = \emptyset$, respectively, for all $j > p$.

ρ' is obviously a run of $\mathcal{C}$ on α. To show that ρ' is accepting, we have to show that O_j is empty infinitely many times. Let us assume that this is not the case; that is, let us assume that there is a last element $\rho'(j)$ with $O_j = \emptyset$ and $O_k \neq \emptyset$ for all $k > j$. (Note that $O_{p+1} = \emptyset$ is empty.) Then we have that $i_k = i_{j+1}$ for all $k > j$, and it is easy to show by induction for all $k > j$ that $O_k \times \{k\}$ is the set of states reachable in ${\mathcal{G}_\alpha}^{i_{j+1}}$ from some state in $O_{j+1} \times \{j+1\}$. But since the rank of all states in $O_{j+1} \times \{j+1\}$ is i_{j+1} (and thus even), all of these states are finite in ${\mathcal{G}_\alpha}^{i_{j+1}}$, which implies that there are only finitely many states reachable in ${\mathcal{G}_\alpha}^{i_{j+1}}$ from $O_{j+1} \times \{j+1\}$, and thus contradicts the assumption that $O_k \neq \emptyset$ is non-empty for all $k > j$. ∎

The two lemmata of this subsection immediately imply the claimed language complementation:

Corollary 3.3. *For a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, the automaton $\mathcal{C}$ resulting from the construction introduced in Subsection 3.1 recognises the complement language of $\mathcal{A}$. ($\mathcal{L}(\mathcal{C}) = \overline{\mathcal{L}(\mathcal{A})}$)* ∎

3.3. Complexity

The costly part in previous approaches [KV01, FKV06] that the proposed method avoids (de facto, although not technically), is a subset construction in addition to the level rankings. Avoiding the subset construction results in a state space reduction by a factor exponential in the size of the automaton $\mathcal{A}$, and even to upper bounds comparable to the established lower bounds [Yan08].

The subset construction is de facto avoided, because it can be encoded into the ranking function once we allow for a slightly enlarged set of output values. For example, we could map all states not in S to -1, and all states in O to -2. Following this convention, the first two elements of every tuple in Q_2 could be pruned. (They remain explicit in the construction because this representation is more comprehensible, and outlines the connection to the older constructions of Friedgut, Kupferman, and Vardi [KV01, FKV06].

Theorem 3.4. *For a given nondeterministic Büchi automaton $\mathcal{A}$ with n states, the automaton $\mathcal{C}$ has $O\big(\mathit{tight}(n+1)\big)$ states.*

Proof. Q_1 is obtained by a simple subset construction, and hence $Q_1 \in O(2^n)$, which is a small subset of $O\big(\mathit{tight}(n+1)\big)$. For a fixed i, a state (S, O, f, i) with an S-tight level ranking f of rank r can be represented by a function $g : Q \to \{-2, -1, \ldots, r\}$ that maps every state $q \in O$ to $g(q) = -1$, every state $q \in Q \setminus S$ not in S to $g(q) = -2$, and every other state $q \in S \setminus O$ to $g(q) = f(q)$. Every such function g either has a domain $g(Q) \subseteq \{0, 1, \ldots, r\}$ and is onto $\{1, 3, \ldots, r\}$, or is a function to $\{-2, -1, \ldots, r\}$ and onto $\{-1\} \cup \{1, 3, \ldots, r\}$ or $\{-2\} \cup \{1, 3, \ldots, r\}$. The size of all three groups of functions is hence in $O\big(\mathit{tight}(n)\big)$ (for a fixed i), which results in an overall size in $O\big(\mathit{tight}(n+1)\big) = O\big(n \cdot \mathit{tight}(n)\big)$. ∎

Together with Proposition 2.4, this establishes tight bounds for Büchi complementation:

Corollary 3.5. *The minimal size of the state space of a nondeterministic Büchi automaton that accept the complement language of a nondeterministic Büchi automaton with n states is in $\Omega\big(\mathit{tight}(n-1)\big)$ and $O\big(\mathit{tight}(n+1)\big)$.* ∎

4. Reduced Average Outdegree

A flaw in the construction presented in Section 3 is that it is optimal only with respect to the state space of the automata. In [KV01], Kupferman and Vardi discuss how to reduce the number of edges such that the bound on the number of edges becomes trilinear in the alphabet size, the bound on the number of states, and the rank of the resulting automaton (see also [GKSV03]). In this section we improve the construction from the previous section such that the bound on the number of edges is merely bilinear in the bound on the number of states and the size of the input alphabet. The technique can be adapted to generally restrict the outdegree to $|\delta(q,\sigma)| \leq 2$ when level rankings are not required to be tight.

4.1. Construction

The automaton $\mathcal{C}$ obtained from the construction described in Section 3 operates in two phases. In a first phase, it stays in Q_1 and only tracks the reachable states of the Büchi automata $\mathcal{A}$ it complements. It then guesses a point $p \in \omega$ such that all levels $j > p$ of $\mathcal{G}_\alpha$ have a tight level ranking to transfer to Q_2.

We improve over the construction of Section 3 by restricting the number of entry points to Q_2 from $O\big(\mathit{tight}(n)\big)$ to $O(n!)$, and by restricting the number of outgoing transitions $|\delta(q,\sigma)| \leq 2$ for all states $q \in Q_2$ and input letters $\sigma \in \Sigma$ to two. The latter is achieved by allowing only the successor $(S,O,f,i) \in \delta_3(q,\sigma)$ with a point wise maximal function f (the γ_3-transitions) or with a function f that is maximal among the final states $(S,O,f,i) \in \delta_3(q,\sigma) \cap F$ among them (the γ_4-transitions). If such elements exist, then they are unique.

The first restriction is achieved by restricting δ_2 to states (S,O,f,i) for which f is maximal with respect to S. We call an S-tight level ranking f with rank r *maximal with respect to* S if it maps all final states $q \in F \cap S$ in S to $r-1$, exactly one state to every odd number $o < r$ smaller than r ($|f^{-1}(o)| = 1$) and all remaining states of S to r, and denote the set of tight rankings that are maximal with respect to S by $\mathcal{M}_S = \{f \in \mathcal{T} \mid f \text{ is maximal with respect to } S\}$.

As there are only $|Q_1| \leq 2^n$ states in Q_1, the impact of their high outdegree is outweighed by the small outdegree ($|\delta(q,\sigma)| \leq 2$) of the remaining $|Q_2| \in O\big(\mathit{tight}(n+1)\big)$ states.

Construction: For a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$ with $n = |Q|$ states, let $\mathcal{D} = (\Sigma, Q', I', \gamma, F')$ denote the nondeterministic Büchi automaton with Q', I', F', Q_1, Q_2, and δ_1 as in the construction from Section 3, and with $\gamma = \delta_1 \cup \gamma_2 \cup \gamma_3 \cup \gamma_4$ for

- $\gamma_2 : Q_1 \times \Sigma \to 2^{Q_2}$ with $(S',O,g,i) \in \gamma_2(S,\sigma) \Leftrightarrow (S',O,g,i) \in \delta_2(S,\sigma)$ and $g \in \mathcal{M}_{S'}$,
- $\gamma_3 : Q_2 \times \Sigma \to 2^{Q_2}$ with $\gamma_3\big((S,O,f,i),\sigma\big) = \big\{\mathit{max}_g\big\{(S',O',g,i') \in \delta_3\big((S,O,f,i),\sigma\big)\big\}\big\}$,
- $\gamma_4 : Q_2 \times \Sigma \to 2^{Q_2}$ with $(S',O'',g',i') \in \gamma_4\big((S,O,f,i),\sigma\big)$ if
  $(S',O',g,i') \in \gamma_3\big((S,O,f,i),\sigma\big)$, $O'' = \emptyset$, $i' \neq 0 \vee O' = \emptyset$, and
  $g'(q) = g(q) - 1$ for all $q \in O'$ and $g'(q) = g(q)$ otherwise,

where $\mathit{max}_g\big\{(S',O',g,i') \in \delta_3\big((S,O,f,i),\sigma\big)\big\}$ selects the unique element with a (point wise) maximal function g. The supremum over all function obviously exists, but it is not necessarily tight. (In this case, max_g returns the empty set.) Since δ_3 is strict with respect to the selection of the other three elements S', O' and i' for a fixed ranking function, the mapping of γ_3 consists only of singletons and the empty set.

While γ_3 selects a maximal successor, γ_4 selects a maximal final successor, which only requires to decrease the value assigned to the states in O' by the ranking function g by one.

(Which cannot be done if their value is already 0, hence the restriction $i \neq 0$ or $O' = \emptyset$.) The tightness of g' is then inherited from the tightness of g.

4.2. Correctness

While it is clear that the language of $\mathcal{D}$ is contained in the language of $\mathcal{C}$, the converse is less obvious. To prove $\mathcal{L}(\mathcal{C}) \subseteq \mathcal{L}(\mathcal{D})$, we show that any ω-word α rejected by $\mathcal{A}$ will be accepted by $\mathcal{D}$ by exploiting the "standard" run ρ' of $\mathcal{C}$ on α from the proof of Lemma 3.2 to build an accepting run ρ'' of $\mathcal{D}$ on α.

Proposition 4.1. *For a given nondeterministic Büchi automaton $\mathcal{A} = (\Sigma, Q, I, \delta, F)$, the automaton $\mathcal{D}$ resulting from the construction introduced in Subsection 4.1 accepts an ω-word $\alpha : \omega \to \Sigma$ if and only if α is rejected by $\mathcal{A}$. ($\mathcal{L}(\mathcal{D}) = \overline{\mathcal{L}(\mathcal{A})}$)*

Proof. We show $\mathcal{L}(\mathcal{D}) = \overline{\mathcal{L}(\mathcal{A})}$ by demonstrating $\overline{\mathcal{L}(\mathcal{A})} \subseteq \mathcal{L}(\mathcal{D}) \subseteq \mathcal{L}(\mathcal{C}) \subseteq \overline{\mathcal{L}(\mathcal{A})}$, where the second inclusion is implied by the fact that every (accepting) run of $\mathcal{D}$ is also an (accepting) run of $\mathcal{C}$, and the third inclusion is shown in Lemma 3.1.

To demonstrate $\overline{\mathcal{L}(\mathcal{A})} \subseteq \mathcal{L}(\mathcal{D})$, we reuse the proof of Lemma 3.2 to obtain an accepting run $\rho' = S_0, S_1, S_2, \ldots, S_p, (S_{p+1}, O_{p+1}, f_{p+1}, i_{p+1}), (S_{p+2}, O_{p+2}, f_{p+2}, i_{p+2}), \ldots$ for $\mathcal{C}$, where $f_j(q)$ is the rank of (q, j) in $\mathcal{G}_\alpha$ for all $j > p$ and $q \in S_j$. (If $\mathcal{A}$ has no run on α, then $\mathcal{D}$ has the same accepting standard run on α that stays in Q_1 as $\mathcal{C}$.)

Let us pick an S_{p+1}-tight ranking function g_{p+1} with the same rank as f_{p+1} that is maximal with respect to S_{p+1}. We show that we then can construct the run

$$\rho'' = S_0, S_1, S_2, \ldots, S_p, (S_{p+1}, O'_{p+1}, g_{p+1}, i'_{p+1}), (S_{p+2}, O'_{p+2}, g_{p+2}, i'_{p+2}), \ldots$$

of $\mathcal{D}$ on α that satisfies $O_{p+1} = \emptyset$, and $i_{p+1} = 0$ and, for all $j > p$,

- $(S_{j+1}, O'_{j+1}, g_{j+1}, i'_{j+1}) \in \gamma_4\big((S_j, O'_j, g_j, i'_j), \alpha(j)\big)$ if $O_{j+1} = \emptyset$ and $i_{j+1} = i'_{j+1}$ (note that i'_{j+1} does not depend on taking the transition from γ_3 or γ_4), and
- $(S_{j+1}, O'_{j+1}, g_{j+1}, i'_{j+1}) \in \gamma_3\big((S_j, O'_j, g_j, i'_j), \alpha(j)\big)$ otherwise.

To show by induction that $g_j \geq f_j$ holds true for all $j > p$ (where $\geq$ is the point wise comparison), we strengthen the claim by claiming additionally that if, for some position $k > p$, $\rho'(k)$ is a final state, $i'_{k+1} = i_{k+1}$ holds true, and $k' > k$ is the next position for which $\rho''(k')$ is final, then $q \in O'_j$ implies $q \in O_j \vee g_j(q) > f_j(q)$ for all $k < j \leq k'$.

For $j = p + 1$ this holds trivially (basis). For the induction step, let us first consider the case of γ_3-transitions. Then $g_{j+1} \geq f_{j+1}$ is implied, because g_{j+1} is maximal among the S_{j+1}-tight level rankings $\leq^{S_j}_{\alpha(j)} g_j$, and $g_j \geq f_j$ holds by induction hypothesis. If $\rho'(j)$ is a final state and $i'_{j+1} = i_{j+1}$ holds true, then $O_{j+1} = {f_j}^{-1}(i_{j+1})$, and hence $q \in O'_j$ implies $g_{j+1}(q) = i'_{j+1} = i_{j+1}$, which implies $q \in O_{j+1} \vee g_{j+1}(q) > f_{j+1}(q)$ (using $g_{j+1} \geq f_{j+1}$).

If a γ_4-transition is taken, then taking a γ_3-transition implied $g_{j+1} \geq f_{j+1}$ and $g_{j+1}(q) = i'_{j+1} = i_{j+1} \Rightarrow g_{j+1}(q) > f_{j+1}(q)$ (note that O_{j+1} is empty) by the previous argument. This immediately implies $g_{j+1} \geq f_{j+1}$ for the γ_4-transition. Consequently, $O_{j+1} = {f_j}^{-1}(i_{j+1})$ (which holds as $\rho'(j)$ is final) entails that $q \in O'_j$ implies $q \in O_{j+1} \vee g_{j+1}(q) > f_{j+1}(q)$.

It remains to show that all functions g_j are S_j-tight level rankings. To demonstrate this, let $q \in S_{p+1}$ be a state of the automaton $\mathcal{A}$ such that $g_{p+1}(q) = o$ is the rank of $(q, p+1)$ in $\mathcal{G}_\alpha$ for an odd number $o \leq r$. (Such a state exists for every odd number $o \leq r$ by construction.) Since $(q, p+1)$ is endangered but not finite in ${\mathcal{G}_\alpha}^o$, all nodes (q', j) with $j > p$ reachable from $(q, p + 1)$ in ${\mathcal{G}_\alpha}^o$ form an infinite connected sub-DAG of ${\mathcal{G}_\alpha}^o$, all of whose

nodes have rank o. (Which, by the proof in Lemma 3.2, entails that $f_j(q') = o$ holds for every vertex (q', j) of this sub-DAG.) By definition of ρ'', it is easy to show by induction that $g_j(q') \leq o$ holds for all of these nodes (q', j). As we have just demonstrated $g_j(q') \geq f_j(q')$, this entails $g_j(q') = o$. Since o can be any odd number less or equal to the rank r of $\mathcal{G}_\alpha$, and since there is, for every $j > p$, some vertex (q', j) reachable from $(q, p+1)$ in ${\mathcal{G}_\alpha}^o$, g_j is an S_j-tight level ranking for all $j > p$.

Finally, the assumption that O'_j is empty only finitely many times implied that there was a last position k such that O'_k is empty. But this implies that i_j is stable for all $j > k$, and within the next n visited fixed points in ρ' there is one that refers to this i_{k+1}. By construction of ρ'', this position is a final state in ρ'', too. $\lightning$ ∎

4.3. Complexity

Extending the tight bound of Section 3 for the state space to a tight bound on the size of the complement automaton is simple: The mappings of δ_1, γ_3, and γ_4 consist of singletons or the empty set, such that only the size of γ_2 needs to be considered more closely.

Theorem 4.2. *For a given nondeterministic Büchi automaton $\mathcal{A}$ with n states and an alphabet of size s, the automaton $\mathcal{D}$ has size $O\big(s\, \mathit{tight}(n+1)\big)$.*

Proof. For all $S \in Q_1$ and $\sigma \in \Sigma$, we have that $\gamma_2(S, \sigma) = \{S'\} \times \{\emptyset\} \times \mathcal{M}_{S'} \times \{0\}$ for $S' = \delta(S)$. Thus, $|\gamma_2(S, \sigma)| = |\mathcal{M}_{S'}|$, which can be estimated by $\sum_{i=1}^{m} \frac{m!}{i!}$ for $m = |S' \smallsetminus F|$, which is in $O(n!)$. Thus $\sum_{S \in Q_1,\ \sigma \in \Sigma} |\gamma_2(S, \sigma)| \in O(s\, 2^n\, n!) \subsetneq o\big(s\, \mathit{tight}(n)\big)$ holds true.

($2^n\, n! \approx \big(\frac{2n}{e}\big)^n \approx (0.74\, n)^n$, whereas $\mathit{tight}(n) \approx (0.76\, n)^n$.) The claim thus follows with Theorem 3.4, and $|\delta_1(q_1, \sigma)| = 1$ and $|\gamma_3(q_2, \sigma)|, |\gamma_4(q_2, \sigma)| \leq 1$ for all $q_1 \in Q_1$, $q_2 \in Q_2$ and $\sigma \in \Sigma$. ∎

Together with Proposition 2.4, this establishes tight complexity bounds for Büchi complementation:

Corollary 4.3. *The complexity of complementing nondeterministic Büchi automata with n states is in $\Omega\big(\mathit{tight}(n-1)\big)$ and $O\big(\mathit{tight}(n+1)\big)$. The discussed complementation technique is therefore optimal modulo a small polynomial factor in $O(n^2)$.* ∎

5. Discussion

This paper marks the end of the long quest for the precise complexity of the Büchi complementation problem. It shows that the previously known lower bound is sharp, which is on one hand surprising, because finding tight lower bounds is generally considered the harder problem, and seems on the other hand natural, because Yan's lower bound builds on the concept of tight level rankings alone [Yan08], while the previously known upper bound [FKV06] incorporates an additional subset construction and builds on estimations on top of this, leaving the estimations of the lower bound the simpler concept of the two.

Similar to the complexity gap in Büchi complementation twenty years ago, the complexity of Büchi determinization is known to be in $n^{\theta(n)}$, but there is also an $n^{\theta(n)}$ gap between the upper [Sch09] and lower [Yan08] bound. Tightening the bounds for Büchi determinization appears to be the natural next step after the introduction of an optimal Büchi complementation algorithm.

References

[Büc62] J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science, 1960, Berkeley, California, USA*, pages 1–11. Stanford University Press, 1962.

[FKV06] Ehud Friedgut, Orna Kupferman, and Moshe Y. Vardi. Büchi complementation made tighter. *International Journal of Foundations of Computer Science*, 17(4):851–867, 2006.

[GKSV03] Sankar Gurumurthy, Orna Kupferman, Fabio Somenzi, and Moshe Y. Vardi. On complementing nondeterministic Büchi automata. In *Proceedings of the 12th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2003), 21–24 October, L'Aquila, Italy*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer-Verlag, 2003.

[Kur94] Robert P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach*. Princeton University Press, 1994.

[KV01] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(2):408–429, July 2001.

[Löd99] Christof Löding. Optimal bounds for transformations of ω-automata. In *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1999), 13–15 December, Chennai, India*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, 1999.

[Mic88] M. Michel. Complementation is more difficult with automata on infinite words. Technical report, CNET, Paris (Manuscript), 1988.

[Péc86] Jean-Pierre Pécuchet. On the complementation of Büchi automata. *Theoretical Computer Science*, 47(3):95–98, 1986.

[Pit07] Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Journal of Logical Methods in Computer Science*, 3(3:5), 2007.

[RS59] Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[Saf88] Shmuel Safra. On the complexity of ω-automata. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS 1988), 24–26 October*, pages 319–327, White Plains, New York, USA, 1988. IEEE Computer Society Press.

[Sch09] Sven Schewe. Tighter bounds for the determinization of Büchi automata. In *Proceedings of the Twelfth International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2009), 22–29 March, York, England, UK*. (to appear), 2009.

[SS78] William J. Sakoda and Michael Sipser. Non-determinism and the size of two-way automata. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC 1978), 1–3 May, San Diego, California, USA*, pages 274–286. ACM Press, 1978.

[SVW87] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(3):217–239, 1987.

[TCT+08] Yih-Kuen Tsay, Yu-Fang Che, Ming-Hsien Tsai, Wen-Chin Chan, and Chi-Jian Luo. GOAL extended: Towards a research tool for omega automata and temporal logic. In *Proceedings of the 14th International Conference On Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), 31 March–3 April, Budapest, Hungary*, volume 4963 of *Lecture Notes in Computer Science*, pages 346–350. Springer-Verlag, 2008.

[Tem93] Nico M. Temme. Asymptotic estimates of Stirling numbers. *Studies in Applied Mathematics*, 89:233–243, 1993.

[Tho99] Wolfgang Thomas. Complementation of Büchi automata revisited. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122. Springer-Verlag, 1999.

[Var07] Moshe Y. Vardi. The Büchi complementation saga. In *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS 2007), 22-24 February, Aachen, Germany*, volume 4393 of *Lecture Notes in Computer Science*, pages 12–22. Springer-Verlag, 2007.

[Yan08] Qiqi Yan. Lower bounds for complementation of *omega*-automata via the full automata technique. *Journal of Logical Methods in Computer Science*, 4(1:5), 2008.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 673–684
www.stacs-conf.org

STRONG COMPLETENESS OF COALGEBRAIC MODAL LOGICS

LUTZ SCHRÖDER [1] AND DIRK PATTINSON [2]

[1] DFKI Bremen and Department of Computer Science, Universität Bremen
E-mail address: Lutz.Schroeder@dfki.de

[2] Department of Computing, Imperial College London
E-mail address: dirk@doc.ic.ac.uk

ABSTRACT. Canonical models are of central importance in modal logic, in particular as they witness strong completeness and hence compactness. While the canonical model construction is well understood for Kripke semantics, non-normal modal logics often present subtle difficulties – up to the point that canonical models may fail to exist, as is the case e.g. in most probabilistic logics. Here, we present a generic canonical model construction in the semantic framework of coalgebraic modal logic, which pinpoints coherence conditions between syntax and semantics of modal logics that guarantee strong completeness. We apply this method to reconstruct canonical model theorems that are either known or folklore, and moreover instantiate our method to obtain new strong completeness results. In particular, we prove strong completeness of graded modal logic with finite multiplicities, and of the modal logic of exact probabilities.

In modal logic, completeness proofs come in two flavours: *weak* completeness, i.e. derivability of all universally valid formulas, is often proved using *finite model* constructions, and *strong* completeness, which additionally allows for a possibly infinite set of assumptions. The latter entails recursive enumerability of the set of consequences of a recursively enumerable set of assumptions, and is usually established using (infinite) *canonical models*. The appeal of the first method is that it typically entails decidability. The second method yields a stronger result and has some advantages of its own. First, it applies in some cases where finite models fail to exist, which often means that the logic at hand is undecidable. In such cases, a completeness proof via canonical models will at least salvage recursive enumerability. Second, it allows for schematic axiomatisations, e.g. pertaining to the infinite evolution of a system or to observational equivalence, i.e. statements to the effect that certain states cannot be distinguished by any formula.

In the realm of Kripke semantics, canonical models exist for a large variety of logics and are well understood, see e.g. [2]. But there is more to modal logic than Kripke semantics, and indeed the natural semantic structures used to interpret a large class of modal logics go beyond pure relations. This includes e.g. the selection function semantics of conditional logics [4], the semantics

1998 ACM Subject Classification: F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic — modal logic; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods — modal logic, representation languages.

Key words and phrases: Logic in computer science, semantics, deduction, modal logic, coalgebra.

Work of the first author performed as part of the DFG project *Generic Algorithms and Complexity Bounds in Coalgebraic Modal Logic* (SCHR 1118/5-1). Work of the second author partially supported by EPSRC grant EP/F031173/1.

of probabilistic logics in terms of probability distributions, and the game frame semantics of coalition logic [16]. To date, there is very little research that provides systematic criteria, or at least a methodology, for establishing strong completeness for logics not amenable to Kripke semantics. This is made worse as the question of strong completeness crucially depends on the chosen semantic domain, which as illustrated above may differ widely. It is precisely this variety in semantics that makes it hard to employ the strong-completeness-via-canonicity approach, as in many cases there is no readily available notion of canonical model. The present work improves on this situation by providing a widely applicable generic canonical model construction. More precisely, we establish the existence of quasi-canonical models, that is, models based on the set of maximally consistent sets of formulas that satisfy the truth lemma, as there may be no unique, or canonical, such model in our more general case. In order to cover the large span of semantic structures, we avoid a commitment to a particular class of models, and instead work within the framework of coalgebraic modal logic [15] which precisely provides us with a semantic umbrella for all of the examples above. This is achieved by using coalgebras for an endofunctor T as the semantic domain for modal languages. As we illustrate in examples, the semantics of particular logics is then obtained by particular choices of T. Coalgebraic modal logic serves in particular as a general semantic framework for non-normal modal logics. As such, it improves on neighbourhood semantics in that it retains the full semantic structure of the original models (neighbourhood semantics offers only very little actual semantic structure, and in fact may be regarded as constructed from syntactic material [18]).

In this setting, our criterion can be formulated as a set of coherence conditions that relate the syntactic component of a logic to its coalgebraic semantics, together with a purely semantic condition stating that the endofunctor T that defines the semantics needs to preserve inverse limits weakly, and thus allows for a passage from the finite to the infinite. We are initially concerned with the existence of quasi-canonical models relative to the class of *all* T-coalgebras, that is, whith logics that are axiomatisable by formulas of modal depth uniformly equal to one [17]. As in the classical theory, the corresponding result for logics with extra frame conditions requires that the logic is canonical, i.e. the frame that underlies a quasi-canonical model satisfies the frame conditions, which holds in most cases, but for the time being needs to be established individually for each logic.

Our new criterion is then used to obtain both previously known and novel strong completeness results. In addition to positive results, we dissect a number of logics for which strong completeness fails and show which assumption of our criterion is violated. In particular, this provides a handle on adjusting either the syntax or the semantics of the logic at hand to achieve strong completeness. For example, we demonstrate that the failure of strong completeness for probabilistic modal logic (witnessed e.g. by the set of formulas assigning probability $\geq 1 - 1/n$ to an event for all n but excluding probability 1) disappears in the logic of exact probabilities. Moreover, we show that graded modal logic, and more generally any description logic [1] with qualified number restrictions, role hierarchies, and reflexive, transitive, and symmetric roles, is strongly complete over the multigraph model of [5], which admits infinite multiplicities. While strong completeness fails for the naive restriction of this model to multigraphs allowing only finite multiplicities, we show how to salvage strong completeness using additive (finite-)integer-valued measures. Finally, we prove strong completeness of several conditional logics w.r.t. conditional frames (also known as selection function models); for at least one of these logics, strong completeness was previously unknown.

1. Preliminaries and Notation

Our treatment of strong completeness is parametric in both the syntax and the semantics of a wide range of modal logics. On the syntactic side, we fix a *modal similarity type* Λ consisting of modal

operators with associated arities. Given a similarity type Λ and a countable set P of atomic propositions, the set $\mathcal{F}(\Lambda)$ of Λ-*formulas* is inductively defined by the grammar

$$\mathcal{F}(\Lambda) \ni \phi, \psi ::= p \mid \bot \mid \neg\phi \mid \phi \wedge \psi \mid L(\phi_1, \dots, \phi_n)$$

where $p \in P$ and $L \in \Lambda$ is n-ary; further boolean operators ($\vee$, $\rightarrow$, $\leftrightarrow$, $\top$) are defined as usual. Given any set X (e.g. of formulas, atomic propositions, or sets (!)), we write $\mathsf{Prop}(X)$ for the set of propositional formulas over X and $\Lambda(X) = \{L(x_1, \dots, x_n) \mid L \in \Lambda \text{ is } n\text{-ary}, x_1, \dots, x_n \in X\}$ for the set of formulas arising by applying exactly one operator to elements of X. We instantiate our results to a variety of settings later with the following similarity types:

Examples 1.1. 1. The similarity type Λ_K of standard modal logic consists of a single unary operator $\Box$.

2. Conditional logic [4] is defined over the similarity type $\Lambda_{\mathrm{CL}} = \{\Rightarrow\}$ where the binary operator $\Rightarrow$ is read as a non-monotonic conditional (default, relevant etc.), usually written in infix notation.

3. Graded modal operators [8] appear in expressive description logics [1] in the guise of so-called qualified number restrictions; although we discuss only modal aspects, we use mostly description logic notation and terminology below. The operators of graded modal logic (GML) are $\Lambda_{\mathrm{GML}} = \{(\geq k) \mid k \in \mathbb{N}\}$ with $(\geq k)$ unary. We write $\geq k.\, \phi$ instead of $(\geq k)\phi$. A formula $\geq k.\, \phi$ is read as 'at least k successor states satisfy ϕ', and we abreviate $\Box\phi = \neg \geq 1.\neg\phi$.

4. The similarity type Λ_{PML} of probabilistic modal logic (PML) [14] contains the unary modal operators L_p for $p \in \mathbb{Q} \cap [0, 1]$, read as 'with probability at least p, ...'.

We split axiomatisations of modal logics into two parts: the first group of axioms is responsible for axiomatising the logic w.r.t. the class of *all* (coalgebraic) models, whereas the second consists of frame conditions that impose additional conditions on models. As the class of all coalgebraic models, introduced below, can always be axiomatised by formulas of *rank* 1, i.e. containing exactly one level of modal operators [17] (and conversely, every collection of such axioms admits a complete coalgebraic semantics [18]), we restrict the axioms in the first group accordingly. More formally:

Definition 1.2. A *(modal) logic* is a triple $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ where Λ is a similarity type, $\mathcal{A} \subseteq \mathsf{Prop}(\Lambda(\mathsf{Prop}(P)))$ is a set of *rank-1 axioms*, and $\Theta \subseteq \mathcal{F}(\Lambda)$ is a set of *frame conditions*. We say that $\mathcal{L}$ is a *rank-1 logic* if $\Theta = \emptyset$. If $\phi \in \mathcal{F}(\Lambda)$, we write $\vdash_{\mathcal{L}} \phi$ if ϕ can be derived from $\mathcal{A} \cup \Theta$ with the help of propositional reasoning, uniform substitution, and the congruence rule: from $\phi_1 \leftrightarrow \psi_1, \dots, \phi_n \leftrightarrow \psi_n$ infer $L(\phi_1, \dots, \phi_n) \leftrightarrow L(\psi_1, \dots, \psi_n)$ whenever $L \in \Lambda$ is n-ary. For a set $\Phi \subseteq \mathcal{F}(\Lambda)$ of assumptions, we write $\Phi \vdash_{\mathcal{L}} \phi$ if $\vdash_{\mathcal{L}} \phi_1 \wedge \dots \wedge \phi_n \rightarrow \phi$ for (finitely many) $\phi_1, \dots, \phi_n \in \Phi$. A set Φ is *$\mathcal{L}$-inconsistent* if $\Phi \vdash_{\mathcal{L}} \bot$, and otherwise *$\mathcal{L}$-consistent*.

Examples 1.3. 1. The modal logic K comes about as the rank-1 logic $(\Lambda_K, \mathcal{A}_K, \emptyset)$ where $\mathcal{A}_k = \{\Box\top, \Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)\}$. The logics $K4, S4, KB, \dots$ arise as $(\Lambda_K, \mathcal{A}_K, \Theta)$ where Θ contains the additional axioms that define the respective logic [2], e.g. $\Theta = \{\Box p \rightarrow \Box\Box p\}$ in the case of $K4$.

2. For conditional logic, we take the similarity type Λ_{CL} together with rank-1 axioms $r \Rightarrow \top$, $r \Rightarrow (p \rightarrow q) \rightarrow ((r \Rightarrow p) \rightarrow (r \Rightarrow q))$ stating that the binary conditional is normal in its second argument. Typical additional rank-1 axioms are

(ID)	$a \Rightarrow a$	*(identity)*
(DIS)	$(a \Rightarrow c) \wedge (b \Rightarrow c) \rightarrow ((a \vee b) \Rightarrow c)$	*(disjunction)*
(CM)	$(a \Rightarrow c) \wedge (a \Rightarrow b) \rightarrow ((a \wedge b) \Rightarrow c)$	*(cautious monotony)*

which together form the so-called *System C*, a modal version of the well-known KLM (Krauss/Lehmannn/Magidor) axioms of default reasoning due to Burgess [3].

3. The axiomatisation of GML given in [8] consists of the rank-1 axioms
$\Box(p \to q) \to (\Box p \to \Box q)$
$\geq k.\, p \to \geq l.\, p$ for $l < k$
$\geq k.\, p \leftrightarrow \bigvee_{i=0,\ldots,k} \geq i.\, (p \wedge q) \wedge \geq (k-i).\, (p \wedge \neg q)$
$\Box(p \to q) \to (\geq k.\, p \to \geq k.\, q)$
Frame conditions of interest include e.g. reflexivity ($p \to \geq 1.\, p$), symmetry ($p \to \Box \geq 1.\, p$), and transitivity ($\geq 1.\, \geq n.\, p \to \geq n.\, p$).

To keep our results parametric also in the semantics of modal logic, we work in the framework of *coalgebraic modal logic* in order to achieve a uniform and coherent presentation. In this framework, the particular shape of models is encapsulated by an endofunctor $T : \mathbf{Set} \to \mathbf{Set}$, the *signature functor* (recall that such a functor maps every set X to a set TX, and every map $f : X \to Y$ to a map $Tf : TX \to TY$ in such a way that composition and identities are preserved), which may be thought of as a parametrised data type. We fix the data Λ, $\mathcal{L}$, T etc. throughout the generic part of the development. The role of models in then played by T-coalgebras:

Definition 1.4. A *T-coalgebra* is a pair $\mathbb{C} = (C, \gamma)$ where C is a set (the *state space* of $\mathbb{C}$) and $\gamma : C \to TC$ is a function, the transition structure of $\mathbb{C}$.

We think of TC as a type of successors, polymorphic in C. The transition structure γ associates a structured collection of successors $\gamma(c)$ to each state $x \in C$. The following choices of signature functors give rise to the semantics of the modal logics discussed in Expl. 1.3.

Examples 1.5. 1. Coalgebras for the covariant powerset functor $\mathcal{P}$ defined on sets X by $\mathcal{P}(X) = \{A \mid A \subseteq X\}$ and on maps f by $\mathcal{P}(f)(A) = f[A]$ are Kripke frames, as relations $R \subseteq W \times W$ on a set W of worlds are in bijection with functions of type $W \to \mathcal{P}(W)$. Restricting the powerset functor to *finite* subsets, i.e. putting $\mathcal{P}_\omega(X) = \{A \subseteq X \mid A \text{ finite}\}$, one obtains the class of image finite Kripke frames as $\mathcal{P}_\omega$-coalgebras.

2. The semantics of conditional logic is captured coalgebraically by the endofunctor $\mathcal{S}$ that maps a set X to the set $(\mathcal{P}(X) \to \mathcal{P}(X))$ of selection functions over X (the action of $\mathcal{S}$ on functions $f : X \to Y$ is given by $\mathcal{S}(f)(s)(B) = f[s(f^{-1}[B])]$). The ensuing $\mathcal{S}$-coalgebras are precisely the conditional frames of [4].

3. The *(infinite) multiset functor* $\mathcal{B}_\infty$ maps a set X to the set $\mathcal{B}_\infty X$ of multisets over X, i.e. functions of type $X \to \mathbb{N} \cup \{\infty\}$. Accordingly, $\mathcal{B}_\infty$-coalgebras are *multigraphs* (graphs with edges annotated by multiplicities). Multigraphs provide an alternative semantics for GML which is in many respects more natural than the original Kripke semantics [5], as also confirmed by new results below.

4. Finally, if $\mathrm{supp}(\mu) = \{x \in X \mid \mu(x) \neq 0\}$ is the support of a function $\mu : X \to [0,1]$ and $\mathcal{D}(X) = \{\mu : X \to [0,1] \mid \mathrm{supp}(\mu) \text{ finite}, \sum_{x \in X} \mu(x) = 1\}$ is the set of finitely supported probability distributions on X, then $\mathcal{D}$-coagebras are probabilistic transition systems, the semantic domain of PML.

The link between coalgebras and modal languages is provided by predicate liftings [15], which are used to interpret modal operators. Essentially, predicate liftings convert predicates on the state space X into predicates on the set TX of structured collections of states:

Definition 1.6. [15] An *n-ary predicate lifting* ($n \in \mathbb{N}$) for T is a family of maps $\lambda_X : \mathcal{P}X^n \to \mathcal{P}TX$, where X ranges over all sets, satisfying the *naturality* condition

$$\lambda_X(f^{-1}[A_1], \ldots, f^{-1}[A_n]) = (Tf)^{-1}[\lambda_Y(A_1, \ldots, A_n)]$$

for all $f : X \to Y$, $A_1, \ldots, A_n \in \mathcal{P}Y$. (For the categorically minded, λ is a natural transformation $\mathcal{Q}^n \to \mathcal{Q} \circ T^{op}$, where $\mathcal{Q}$ denotes contravariant powerset.) A *structure* for a similarity type Λ over an endofunctor T is the assignment of an n-ary predicate lifting $[\![L]\!]$ to every n-ary modal operator $L \in \Lambda$.

Given a valuation $V : P \to \mathcal{P}(C)$ of the propositional variables and a T-coalgebra (C, γ), a structure for Λ allows us to define a satisfaction relation $\models_{(C,\gamma,V)}$ between states of C and formulas $\phi \in \mathcal{F}(\Lambda)$ by stipulating that $c \models_{(C,\gamma,V)} p$ iff $c \in V(p)$ and

$$c \models_{(C,\gamma,V)} L(\phi_1, \ldots, \phi_n) \text{ iff } \gamma(c) \in [\![L]\!]_C([\![\phi_1]\!], \ldots, [\![\phi_n]\!]),$$

where $[\![\phi]\!] = \{c \in C \mid c \models_{(C,\gamma,V)} \phi\}$. An *$\mathcal{L}$-model* is now a *model*, i.e. a triple (C, γ, V) as above, such that $c \models_{(C,\gamma,V)} \psi$ for all all $c \in C$ and all substitution instances ψ of $\mathcal{A} \cup \Theta$. An *$\mathcal{L}$-frame* is a T-coalgebra (C, γ) such that (C, γ, V) is an $\mathcal{L}$-model for all valuations V. The reader is invited to check that the following predicate liftings induce the standard semantics for the modal languages introduced in Expl. 1.1.

Examples 1.7. 1. A structure for Λ_K over the covariant powerset functor $\mathcal{P}$ is given by $[\![\Box]\!]_X(A) = \{Y \in \mathcal{P}(X) \mid Y \subseteq A\}$. The frame classes defined by the frame conditions mentioned in Expl. 1.3.1 are well-known; e.g. a Kripke frame (X, R) is a $K4$-frame iff R is transitive.

2. Putting $[\![\Rightarrow]\!]_X(A, B) = \{f \in \mathcal{S}(X) \mid f(A) \subseteq B\}$ reconstructs the semantics of conditional logic in a coalgebraic setting.

3. A structure for GML over $\mathcal{B}_\infty$ is given by $[\![(\geq k)]\!]_X(A) = \{f : X \to \mathbb{N} \cup \{\infty\} \mid \sum_{x \in A} f(x) \geq k\}$. The frame conditions mentioned in Expl. 1.3.3 correspond to conditions on multigraphs that can be read off directly from the logical axioms. E.g. a multigraph satisfies the transitivity axiom $\geq 1.\, \geq n.\, p \to \geq n.\, p$ iff whenever x has non-zero transition multiplicity to y and y has transition multiplicity at least n to z, then x has transition multiplicity at least n to z.

4. The structure over $\mathcal{D}$ that captures PML coalgebraically is given by the the predicate lifting $[\![L_p]\!]_X(A) = \{\mu \in \mathcal{D}(X) \mid \sum_{x \in A} \mu(x) \geq p\}$ for $p \in [0, 1] \cap \mathbb{Q}$.

From now on, *fix a modal logic $\mathcal{L} = (\Lambda, \mathcal{A}, \Theta)$ and a structure for Λ over a functor T*. We say that $\mathcal{L}$ is *strongly complete* for some class of models if every $\mathcal{L}$-consistent set of formulas is satisfiable in some state of some model in that class. Restricting to *finite* sets Φ defines the notion of *weak completeness*; many coalgebraic modal logics are only weakly complete [17].

Definition 1.8. Let X be a set. If $\psi \in \mathcal{F}(\Lambda)$ and $\tau : P \to \mathcal{P}(X)$ is a valuation, we write $\psi\tau$ for the result of substituting $\tau(p)$ for p in ψ, with propositional subformulas evaluated according to the boolean algebra structure of $\mathcal{P}(X)$. (Hence, $\psi\tau$ is a formula over the set $\mathcal{P}(X)$ of atoms.) A formula $\phi \in \mathsf{Prop}(\Lambda(\mathcal{P}(X))$ is *one-step $\mathcal{L}$-derivable*, denoted $\vdash^1_{\mathcal{L}} \phi$, if ϕ is propositonally entailed by the set $\{\psi\tau \mid \tau : P \to \mathcal{P}(X), \psi \in \mathcal{A}\}$. A set $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is *one-step $\mathcal{L}$-consistent* if there do not exist formulas $\phi_1, \ldots, \phi_n \in \Phi$ such that $\vdash^1_{\mathcal{L}} \neg(\phi_1 \wedge \cdots \wedge \phi_n)$. Dually, the *one-step semantics* $[\![\phi]\!]^1_X \subseteq TX$ of a formula $\phi \in \mathsf{Prop}(\Lambda(\mathcal{P}(X))$ is defined inductively by $[\![L(A_1, \ldots, A_n)]\!]^1_X = [\![L]\!]_X(A_1, \ldots, A_n)$ for $A_1, \ldots, A_n \subseteq X$. A set $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is *one-step satisfiable* if $\bigcap_{\phi \in \Phi} [\![\phi]\!]^1_X \neq \emptyset$. We say that $\mathcal{L}$ (or Λ) is *separating* if $t \in TX$ is uniquely determined by the set $\{\phi \in \Lambda(\mathcal{P}(X)) \mid t \in [\![\phi]\!]^1_X\}$. We call $\mathcal{L}$ (or $\mathcal{A}$) *one-step sound* if every one-step derivable formula $\phi \in \mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is one-step valid, i.e. $[\![\phi]\!]^1_X = X$.

Henceforth, we assume that $\mathcal{L}$ is one-step sound, so that every T-coalgebra satisfies the rank-1 axioms; in the absence of frame conditions ($\Theta = \emptyset$), this means in particular that every T-coalgebra

is an $\mathcal{L}$-frame. The above notions of one-step satisfiability and one-step consistency are the main concepts employed in the proof of strong completeness in the following section.

Given a structure for Λ over T, every set $\mathcal{B}$ of rank-1 axioms over Λ defines a subfunctor $T_\mathcal{B}$ of $\mathcal{B}$ with $T_\mathcal{B}(X) = \bigcap\{[\![\phi\tau]\!]^1_X \mid \phi \in \mathcal{B}, \tau : P \to \mathcal{P}(X)\} \subseteq TX$. This functor induces a structure for which $\mathcal{B}$ is one-step sound.

Example 1.9. The additional rank-1 axioms of Expl. 1.3.2 induce subfunctors $\mathcal{S}_\mathcal{B}$ of the functor $\mathcal{S}$ of Expl. 1.5.2. E.g. we have

$$\mathcal{S}_{\{ID\}}X = \{f \in \mathcal{S}(X) \mid \forall A \subseteq X.\, f(A) \subseteq A\}$$
$$\mathcal{S}_{\{ID,DIS\}}X = \{f \in \mathcal{S}(X) \mid \forall A, B \subseteq X.\, f(A) \subseteq A \wedge f(A \cup B) \subseteq f(A) \cup f(B)\}$$
$$\mathcal{S}_{\{ID,DIS,CM\}}X = \{f \in \mathcal{S}(X) \mid \forall A, B \subseteq X.\, f(A) \subseteq A \wedge (f(B) \subseteq A \Rightarrow f(A) \cap B \subseteq f(B))\}$$

(it is an amusing exercise to verify the last claim).

2. Strong Completeness Via Quasi-Canonical Models

We wish to establish strong completeness of $\mathcal{L}$ by defining a suitable T-coalgebra structure ζ on the set S of maximally $\mathcal{L}$-consistent subsets of $\mathcal{F}(\Lambda)$, equipped with the standard valuation $V(p) = \{\Gamma \in S \mid p \in \Gamma\}$. The crucial property required is that ζ be *coherent*, i.e.

$$\zeta(\Gamma) \in [\![L]\!](\hat{\phi}_1, \ldots, \hat{\phi}_n) \iff L(\phi_1, \ldots, \phi_n) \in \Gamma,$$

where $\hat{\phi} = \{\Delta \in S \mid \phi \in \Delta\}$, for $L \in \Lambda$ n-ary, $\Gamma \in S$, and $\phi_1, \ldots, \phi_n \in \mathcal{F}(\Lambda)$, as this allows proving, by a simple induction over the structure of formulas,

Lemma 2.1 (Truth lemma)**.** *If ζ is coherent, then for all formulas ϕ, $\Gamma \models_{(S,\zeta,V)} \phi$ iff $\phi \in \Gamma$.*

We define a *quasi-canonical model* to be a model (S, ζ, V) with ζ coherent; the term quasi-canonical serves to emphasise that the coherence condition does not determine the transition structure ζ uniquely. By the truth lemma, quasi-canonical models for $\mathcal{L}$ are $\mathcal{L}$-models, i.e. satisfy all substitution instances of the frame conditions. The first question is now under which circumstances quasi-canonical models exist; we proceed to establish a widely applicable criterion. This criterion has two main aspects: a *local* form of strong completeness involving only finite sets, and a preservation condition on the functor enabling passage from finite sets to certain infinite sets. We begin with the latter part:

Definition 2.2. A *surjective ω-cochain (of finite sets)* is a sequence $(X_n)_{n\in\mathbb{N}}$ of (finite) sets equipped with surjective functions $p_n : X_{n+1} \to X_n$ called *projections*. The *inverse limit* $\varprojlim X_n$ of (X_n) is the set $\{(x_i) \in \prod_{i\in\mathbb{N}} X_i \mid \forall n.\, p_n(x_{n+1}) = x_n\}$ of *coherent* families (x_i). The *limit projections* are the maps $\pi_i((x_n)_{n\in\mathbb{N}}) = x_i$, $i \in \mathbb{N}$; note that the π_i are surjective, i.e. every $x \in X_i$ can be extended to a coherent family. Since all set functors preserve surjections, (TX_n) is a surjective ω-cochain with projections Tp_n. The functor T *weakly preserves inverse limits of surjective ω-cochains of finite sets* if for every surjective ω-cochain (X_n) of finite sets, the canonical map $T(\varprojlim X_n) \to \varprojlim TX_n$ is surjective, i.e. every coherent family (t_n) in $\prod TX_n$ is *induced* by a (not necessarily unique) $t \in T(\varprojlim X_n)$ in the sense that $T\pi_n(t) = t_n$ for all n.

Example 2.3. Let A be a finite alphabet; then the sets A^n, $n \in \mathbb{N}$, form a surjective ω-cochain of finite sets with projections $p_n : A^{n+1} \to A^n$, $(a_1, \ldots, a_{n+1}) \mapsto (a_1, \ldots, a_n)$. The inverse limit $\varprojlim A^n$ is the set A^ω of infinite sequences over A. The covariant powerset functor $\mathcal{P}$ preserves this inverse limit weakly: given a coherent family of subsets $B_n \subseteq A^n$, i.e. $p_n[B_{n+1}] = B_n$ for all n,

we define the set $B \subseteq A^\omega$ as the set of all infinite sequences $(a_n)_{n\geq 1}$ such that $(a_1, \ldots, a_n) \in B_n$ for all n; it is easy to check that indeed B induces the B_n, i.e. $\pi_n[B] = B_n$. However, B is by no means uniquely determined by this property: Observe that B as just defined is a safety property. The intersection of B with any liveness property C, e.g. the set C of all infinite sequences containing infinitely many occurrences of a fixed letter in A, will also satisfy $\pi_n[B \cap C] = B_n$ for all n.

The second part of our criterion is an infinitary version of a local completeness property called one-step completeness, which has been used previously in *weak* completeness proofs [15, 17].

Definition 2.4. We say that $\mathcal{L}$ is *strongly one-step complete over finite sets* if for finite X, every one-step consistent subset Φ of $\mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is one-step satisfiable.

The difference with plain one-step completeness is that Φ above may be infinite. Consequently, strong and plain one-step completeness coincide in case the modal similarity type Λ is finite, since in this case, $\mathsf{Prop}(\Lambda(\mathcal{P}(X)))$ is, for finite X, finite up to propositional equivalence. The announced strong completeness criterion is now the following.

Theorem 2.5. *If $\mathcal{L}$ is strongly one-step complete over finite sets and separating, Λ is countable, and T weakly preserves inverse limits of surjective ω-cochains of finite sets, then $\mathcal{L}$ has a quasi-canonical model.*

Proof sketch. The most natural argument is via the dual adjunction between sets and boolean algebras that associates to a set the boolean algebra of its subsets, and to a boolean algebra the set of its ultrafilters. For economy of presentation, we outline a direct proof instead: we prove that

($*$) every maximally one-step consistent $\Phi \subseteq \mathsf{Prop}(\Lambda(\mathfrak{A}))$ is one-step satisfiable, where $\mathfrak{A} = \{\hat{\phi} \mid \phi \in \mathcal{F}(\Lambda)\} \subseteq \mathcal{P}(S)$.

The existence of the required coherent coalgebra structure ζ on S follows immediately, since the coherence requirement for $\zeta(\Gamma)$, $\Gamma \in S$, amounts to one-step satisfaction of a maximally one-step consistent subset of $\mathsf{Prop}(\Lambda(\mathfrak{A}))$.

To prove ($*$), let $\Lambda = \{L_n \mid n \in \mathbb{N}\}$, let $P = \{p_n \mid n \in \mathbb{N}\}$, let $\mathcal{F}_n$ denote the set of Λ-formulas of modal nesting depth at most n that employ only modal operators from $\Lambda_n = \{L_0, \ldots, L_n\}$ and only the atomic propositions $p_0, \ldots, p_n$, and let S_n be the set of maximally consistent subsets of $\mathcal{F}_n$. Then S is (isomorphic to) the inverse limit $\varprojlim S_n$, where the projections $S_{n+1} \to S_n$ and the limit projections $S \to S_n$ are just intersection with $\mathcal{F}_n$. As the sets S_n are finite, we obtain by strong one-step completeness $t_n \in TS_n$ such that $t_n \models^1_{S_n} \Phi \cap \mathsf{Prop}(\Lambda(\mathfrak{A}_n))$, where $\mathfrak{A}_n = \{\hat{\phi} \cap S_n \mid \phi \in \mathcal{F}_n\}$. By separation, $(t_n)_{n\in\mathbb{N}}$ is coherent, and hence is induced by some $t \in TS$ by weak preservation of inverse limits; then, $t \models^1_S \Phi$. ∎

Together with the Lindenbaum Lemma we obtain strong completeness as a corollary.

Corollary 2.6. *Under the conditions of Thm. 2.5, $\mathcal{L}$ is strongly complete for $\mathcal{L}$-models.*

Both Thm. 2.5 and Cor. 2.6 do apply to the case that $\mathcal{L}$ has frame conditions. When $\mathcal{L}$ is of rank 1 (i.e. $\Theta = \emptyset$), Cor. 2.6 implies that $\mathcal{L}$ is strongly complete for (models based on) $\mathcal{L}$-frames. In the presence of frame conditions, the underlying frame of an $\mathcal{L}$-model need not be an $\mathcal{L}$-frame, so that the question arises whether $\mathcal{L}$ is also strongly complete for $\mathcal{L}$-frames. In applications, positive answers to this question, usually referred to as the canonicity problem, typically rely on a judicious choice of quasi-canonical model to ensure that the latter is an $\mathcal{L}$-frame, often the largest quasi-canonical model under some ordering on TS. Detailed examples are given in Sec. 3.

Remark 2.7. It is shown in [13] that T admits a strongly complete modal logic if T weakly preserves (arbitrary) inverse limits *and preserves finite sets*. The essential contribution of the above

result is to remove the latter restriction, which fails in important examples. Moreover, the observation that we need only consider *surjective* ω-cochains is relevant in some applications, see below.

Remark 2.8. A last point that needs clearing up is whether strong completeness of coalgebraic modal logics can be established by some more general method than quasi-canonical models of the quite specific shape used here. The answer is negative, at least in the case of rank-1 logics $\mathcal{L}$: it has been shown in [12] that every such $\mathcal{L}$ admits models which consist of the maximally *satisfiable* sets of formulas and obey the truth lemma. Under strong completeness, such models are quasi-canonical.

This seems to contradict the fact that some canonical model constructions in the literature, notably the canonical Kripke models for graded modal logics [8, 6], employ state spaces which have multiple copies of maximally consistent sets. The above argument indicates that such logics fail to be coalgebraic, and indeed this is the case for GML with Kripke semantics. As mentioned above, GML has an alternative coalgebraic semantics over multigraphs, and we show below that this semantics does admit quasi-canonical models in our sense.

3. Examples

We now show how the generic results of the previous section can be applied to obtain canonical models and associated strong completeness and compactness theorems for a large variety of structurally different modal logics. We have included some negative examples where canonical models necessarily fail to exist due to non-compactness, and we analyse which conditions of Thm. 2.5 fail in each case. We emphasise that in the positive examples, the verification of said conditions is entirely stereotypical. Weak preservation of inverse limits of surjective ω-cochains usually holds without the finiteness assumption, which is therefore typically omitted.

Example 3.1 (Strong completeness of Kripke semantics for K)**.** Recall from Expl. 1.5.1 that Kripke frames are coalgebras for the powerset functor $TX = \mathcal{P}(X)$. Strong completeness of K with respect to Kripke semantics is, of course, well known. We briefly illustrate how this can be derived from our coalgebraic treatment. To see that K is strongly one-step complete over finite sets X, let $\Phi \subseteq \mathsf{Prop}(\Lambda_K(\mathcal{P}(X)))$ be maximally one-step consistent. It is easy to check that $\{x \in X \mid \Diamond\{x\} \in \Phi\}$ satisfies Φ. To prove that the powerset functor weakly preserves inverse limits, let (X_n) be an ω-cochain, and let $(A_n \in \mathcal{P}(X_n))$ be a coherent family. Then (A_n) is itself a cochain, and the set $A = \varprojlim A_n \subseteq \varprojlim X_n$ induces (A_n) (w.r.t. the subset ordering on $\mathcal{P}(X)$). Separation is clear. By Thm. 2.5, there exists a quasi-canonical Kripke model for all normal modal logics. In particular, the standard canonical model [4] is quasi-canonical; it witnesses strong completeness (w.r.t. frames) of all canonical logics such as $K4$, $S4$, $S5$.

Example 3.2 (Failure of strong completeness of K over finitely branching models)**.** As seen in Expl. 1.5.1, finitely branching Kripke frames are coalgebras for the finite powerset functor $\mathcal{P}_\omega$. It is clear that quasi-canonical models fail to exist in this case, as compactness fails over finitely branching frames: one can easily construct formulas ϕ_n that force a state to have at least n different successors. The obstacle to the application of Thm. 2.5 is that the finite powerset functor fails to preserve inverse limits weakly, as the inverse limit of an ω-cochain of finite sets may fail to be finite.

Example 3.3 (Conditional logic)**.** Recall from Expl. 1.5.2 that the conditional logic CK is interpreted over the functor $\mathcal{S}(X) = \mathcal{P}(X) \to \mathcal{P}(X)$. To prove strong one-step completeness over finite sets X, let $\Phi \subseteq \mathsf{Prop}(\Lambda_{\mathrm{CL}}(\mathcal{P}(X)))$ be maximally one-step consistent. Define $f : \mathcal{P}(X) \to \mathcal{P}(X)$ by $f(A) = \bigcap\{B \subseteq X \mid A \Rightarrow B \in \Phi\}$; it is mechanical to check that $f \models^1 \Phi$. To see that $\mathcal{S}$ weakly preserves inverse limits, let (X_n) be a surjective ω-cochain, let $X = \varprojlim X_n$, and let $(f_n \in \mathcal{S}(X_n))$

be coherent. Define $f : \mathcal{P}(X) \to \mathcal{P}(X)$ by letting $(x_n) \in f(A)$ for a coherent family $(x_n) \in X$ iff whenever $A = \pi_n^{-1}[B]$ for some n and some $B \subseteq X_n$, then $x_n \in f_n(B)$. Using surjectivity of the projections of (X_n), it is straightforward to prove that f induces (f_n). Finally, separation is clear. By Thm. 2.5, it follows that the conditional logic CK has a quasi-canonical model, and hence that CK is strongly complete for conditional frames. In the case of the additional rank-1 axioms mentioned in Expl. 1.3.2 and the corresponding subfunctors of $\mathcal{S}$ described in Expl. 1.9, the situation is as follows.

Identity: The functor $\mathcal{S}_{\{ID\}}$ weakly preserves inverse limits of surjective ω-cochains. In the notation above, put $(x_n) \in f(A)$ iff the condition above holds and $(x_n) \in A$.

Identity and disjunction: The functor $\mathcal{S}_{\{ID,DIS\}}$ weakly preserves inverse limits of surjective ω-cochains: put $(x_n) \in f(A)$ iff $(x_n) \in A$ and whenever $(x_n) \in \pi_m^{-1}B \subseteq A$, then $x_m \in f_m(B)$.

System C: It is open whether the the functor $\mathcal{S}_{\{ID,DIS,CM\}}$ weakly preserves inverse limits of surjective ω-cochains, and whether System C is strongly complete over conditional frames.

Indeed it appears to be an open problem to find *any* semantics for which System C is strongly complete, other than the generalised neighbourhood semantics as described e.g. in [18], which is strongly complete for very general reasons but provides little in the way of actual semantic information. The classical preference semantics according to Lewis is only known to be weakly complete [3]. Friedman and Halpern [9] do silently prove strong completeness of System C w.r.t. plausibility measures; however, on close inspection the latter turn out to be essentially equivalent to the above-mentioned generalised neighbourhood semantics. Moreover, Segerberg [19] proves strong completeness for a whole range of conditional logics over *general* conditional frames, where, in analogy to corresponding terminology for Kripke frames, a general conditional frame is equipped with a distinguished set of *admissible propositions* limiting both the range of valuations and the domain of selection functions. In contrast, our method yields full conditional frames in which the frame conditions hold for *any* valuation of the propositional variables. While in the case of CK and its extension by ID alone, these models differ from Segerberg's only in that they insert default values for the selection function on non-admissible propositions, the canonical model for the extension of CK by $\{ID, DIS\}$ has non-trivial structure on non-admissible propositions, and we believe that our strong completeness result for this logic is genuinely new.

Example 3.4 (Strong completeness of GML over multigraphs)**.** Recall from Expl. 1.5.3 that graded modal logic (GML) has a coalgebraic semantics in terms of the multiset functor $\mathcal{B}_\infty$. To prove strong one-step completeness over finite sets X, let $\Phi \subseteq \mathsf{Prop}(\Lambda_{GML}(\mathcal{P}(X)))$ be maximally one-step consistent. We define $B \in \mathcal{B}_\infty(X)$ by $B(A) \geq n \iff \geq n.\, A \in \Phi$; it is easy to check that B is well-defined and additive. To prove weak preservation of inverse limits, let (X_n) be an ω-cochain, let $X = \varprojlim X_n$, and let $(B_n \in \mathcal{B}_\infty(X_n))$ be coherent. Then define $B \in \mathcal{B}_\infty(X)$ pointwise by

$$B((x_n)) = \min_{n \in \mathbb{N}} B_n(x_n),$$

noting that the sequence $(B_n(x_n))$ is decreasing by coherence. A straightforward computation shows that B induces (B_n). Separation is clear.

By the above and Thm. 2.5, all extensions of GML have quasi-canonical multigraph models. While the technical core of the construction is implicit in the work of Fine [8] and de Caro [6], these authors were yet unaware of multigraph semantics, and hence our result that *GML is strongly complete over multigraphs* has not been obtained previously.

The standard frame conditions for reflexivity, symmetry, and transitivity (Expls. 1.5.3 and 1.7. 3) and arbitrary combinations thereof are easily seen to be satisfied in the quasi-canonical model constructed above. We point out that this contrasts with Kripke semantics in the case of the

graded version of $S4$, i.e. GML extended with the reflexivity and transitivity axioms of Expl. 1.5.3: as shown in [7], the complete axiomatisation of graded modal logic over transitive reflexive Kripke frames includes two rather strange combinatorial artefacts, which by the above disappear in the multigraph semantics. The reason for the divergence (which we regard as an argument in favour of multigraph semantics) is that, while in many cases multigraph models are easily transformed into equivalent Kripke models by just making copies of states, no such translation exists in the transitive reflexive case (transitivity alone is unproblematic).

Observe moreover that the above extends straightforwardly to decription logics $\mathcal{ALCQ}(\mathcal{R})$ with qualified number restrictions and a role hierarchy $\mathcal{R}$ where roles may be distinguished as, in any combination, transitive, reflexive, or symmetric. As shown in [10, 11], $\mathcal{ALCQ}(\mathcal{R})$ is undecidable for many $\mathcal{R}$, even when only transitive roles are considered. For undecidable logics, completeness is in some sense the 'next best thing', as it guarantees if not recursiveness then at least recursive enumerability of all valid formulas, and hence enables automatic reasoning. Essentially, our results show that the natural axiomatisation of $\mathcal{ALCQ}(\mathcal{R})$ *with transitive, symmetric and reflexive roles is strongly complete over multigraphs*, a result which fails for the standard Kripke semantics.

Example 3.5 (Failure of strong completeness of image-finite GML)**.** Similarly to the case of image-finite Kripke frames, one can model an image-finite version of graded modal logic coalgebraically by exchanging the functor $\mathcal{B}_\infty$ for the *finite multiset functor* $\mathcal{B}$, where $\mathcal{B}(X)$ consists of all maps $X \to \mathbb{N}$ with finite support. Of course, the resulting logic is non-compact and hence fails to admit a canonical model. This is witnessed not only by the same family of formulas as in the case of image-finite Kripke semantics, which targets finiteness of the number of different successors, but also by the set of formulas $\{\geq n.\, a \mid n \in \mathbb{N}\}$, which targets finiteness of multiplicities. Analysing the conditions of Thm. 2.5, we detect two violations: not only does weak preservation of inverse limits fail, but there is also no way to find an axiomatisation which is strongly one-step complete over finite sets (again, consider sets $\{\geq n.\, \{x\} \mid n \in \mathbb{N}\}$).

Strong completeness of image-finite GML can be recovered by slight adjustments to the syntax and semantics. We formulate a more general approach, as follows.

Example 3.6 (Strong completeness of the logic of additive measures)**.** We fix an at most countable commutative monoid M (e.g. $M = \mathbb{N}$). We think of the elements of M as describing the measure of a set of elements. To ensure compactness, we have to allow some sets to have undefined measure. That is, we work with coalgebras for the endofunctor T_M defined by

$$T_M(X) = \{(\mathfrak{A}, \mu) \mid \mathfrak{A} \subseteq \mathcal{P}(X) \text{ closed under disjoint unions}, \mu : \mathfrak{A} \to M \text{ additive}\}$$

The modal logic of additive M-valued measures is given by the similarity type $\Lambda_M = \{E_m \mid m \in M\}$ where $E_m\phi$ expresses that ϕ has measure m, i.e.

$$[\![E_m]\!]_X B = \{(\mathfrak{A}, \mu) \in T_M(X) \mid B \in \mathfrak{A}, \mu(B) = m\}.$$

Λ_M is clearly separating. The logic is axiomatised by the following two axioms:

$$E_m a \to \neg E_n a \quad (n \neq m) \quad \text{and} \quad E_m(a \wedge b) \wedge E_n(a \wedge \neg b) \to E_{m+n} a.$$

These axioms are strongly one-step complete over finite sets X: if $\Phi \subseteq \mathsf{Prop}(\Lambda_M(\mathcal{P}(X)))$ is maximally one-step consistent, then $(\mathfrak{A}, \mu) \models^1 \Phi$ where $A \in \mathfrak{A}$ iff $E_m A \in \Phi$ for some necessarily unique m, in which case $\mu(A) = m$. Moreover, T_M weakly preserves inverse limits $X = \varprojlim X_n$, with finite X_n: a coherent family $((\mathfrak{A}_n, \mu_n) \in T_M(X_n))$ is induced by $(\mathfrak{A}, \mu) \in T_M(X)$, where $\mathfrak{A} = \{\pi_n^{-1}[B] \mid n \in \mathbb{N}, B \in \mathfrak{A}_n\}$ and $\mu(\pi_n^{-1}[B]) = \mu_n(B)$ is easily seen to be well-defined and additive. Theorem 2.5 now guarantees existence of quasi-canonical models. A simple example is $M = \mathbb{Z}/2\mathbb{Z}$, which induces a logic of even and odd.

For the case $M = \mathbb{N}$, we obtain a variant of graded modal logic with finite multiplicities, where we code $\geq k.\phi$ as $\neg \bigvee_{0 \leq i < k} E_k \phi$. However, it may still be the case that a state has a family of successor sets of unbounded measure, so that undefinedness of the measure of the entire state space just hides an occurrence of infinity. This defect is repaired by insisting that the measure of the whole state space is finite at the expense of disallowing the modal operator E_0 in the language, as follows.

Example 3.7 (Strong completeness of finitely branching GML^-)**.** To force the entire state space to have finite measure, we additionally introduce a *measurability* operator E, interpreted by $[\![E]\!]B = \{(\mathfrak{A}, \mu) \mid B \in \mathfrak{A}\}$, and impose obvious axioms guaranteeing that measures on X are defined on boolean subalgebras of $\mathcal{P}(X)$, in particular $E\top$ (i.e. $\mu(X)$ is finite), and $E_n a \to Ea$. In order to achieve compactness, we now leave a bolt hole on the syntactical side and exclude the operator E_0. In other words, the syntax of GML^- is given by the similarity type $\Lambda^-_{\mathrm{GML}} = \{E\} \cup \{E_n \mid n > 0\}$, and we interpret GML^- over coalgebras for the functor $\mathcal{B}_M$ defined by

$$\mathcal{B}_M(X) = \{(\mathfrak{A}, \mu) \mid \mathfrak{A} \text{ boolean subalgebra of } \mathcal{P}(X), \mu : \mathfrak{A} \to \mathbb{N} \text{ additive}\}.$$

Separation is clear. The axiomatisation of GML^- is given by the axiomatisation of the modal logic of additive measures, the above-mentioned axioms on E, and the additional axiom

$$E_n a \wedge Eb \to E_n(a \wedge b) \vee E_n(a \wedge \neg b) \vee \bigvee_{0<k<n} (E_k(a \wedge b) \wedge E_{n-k}(a \wedge \neg b))$$

which compensates for the absence of E_0. Strong one-step completeness over finite sets and weak preservation of inverse limits is shown analogously as in Expl. 3.6, so that we obtain a *strongly complete finitely branching graded modal logic* GML^-. The tradeoff is that the operator $\geq k.\phi$ is no longer expressible as $\neg \bigvee_{0 \leq i < k} E_i \phi$ in GML^- which only allows to formulate the implication $\geq 1.\phi \to \geq n.\, \phi$.

Example 3.8 (Failure of strong completeness for PML over finitely supported probability distributions)**.** Like image-finite graded modal logic, probabilistic modal logic as introduced in Expl. 1.5.4 fails to be compact, and violates the conditions of Thm. 2.5 on two counts, namely weak preservation of inverse limits and strong one-step completeness over finite sets. The first issue is related to image-finiteness, while the second is rooted in the structure of the real numbers: e.g. the set $\{L_{1/2-1/n} a \mid n \in \mathbb{N}\} \cup \{\neg L_{1/2} a\}$ is finitely satisfiable but not satisfiable.

Example 3.9 (Strong completeness of the logic of exact probabilities)**.** In order to remove the above-mentioned failure of compactness, we consider the fragment of probabilistic modal logic containing only operators E_p stating that a given event has probability exactly p. (This is, of course, less expressive than the operators L_p but still allows reasonable statements such as that rolling a six on a die happens with probability $1/6$.) Moreover, we require probabilities to be rational and allow probabilities to be undefined, thus following the additive measures approach as outlined above, where we consider a subfunctor of $T_{\mathbb{Q}}$ defined by the requirement that the whole set has measure 1. However, we are able to impose stronger conditions on the domain $\mathfrak{A} \subseteq \mathcal{P}(X)$ of a probability measure P on X: we require that $X \in \mathfrak{A}$ and that $A, B \in \mathfrak{A}$, $B \subseteq A$ imply $A - B \in FA$, which is reflected in the additional axioms $E_1 \top$ and $E_p a \wedge E_q(a \wedge b) \to E_{p-q}(a \wedge \neg b)$. It is natural that we cannot force closure under intersection, as there is in general no way to infer the exact probability of $A \cap B$ from the probabilities of A and B. Along the same lines as above, we now obtain quasi-canonical models, and hence strong completeness and compactness, of the arising modal logic of exact probabilities.

4. Conclusion

We have laid out a systematic method of proving existence of canonical models in a generic semantic framework encompassing a wide range of structurally different modal logics. We have shown how this method turns the construction of canonical models into an entirely mechanical exercise where applicable, and points the way to obtaining compact fragments of non-compact logics. As example applications, we have reproved a number of known strong completeness result and established several new results of this kind; specifically, the latter includes strong completeness of the following logics.

- The modal logic of exact probabilities, with operators E_p 'with probability exactly p'.
- Graded modal logic over transitive reflexive multigraphs, i.e. the natural graded version of $S4$, and more generally description logic with role hierarchies including transitive, reflexive, and symmetric roles and qualified number restrictions also on non-simple (e.g. transitive) roles.
- The conditional logic $CK + \{ID, DIS\}$, i.e. with the standard axioms of identity and disjunction, interpreted over conditional frames.

A number of interesting open problems remain, e.g. to find further strongly complete variants of probabilistic modal logic or to establish strong completeness of the full set of standard axioms of default logic, Burgess' System C [3], over the corresponding class of conditional frames.

References

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook*. Cambridge University Press, 2003.
[2] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
[3] J. Burgess. Quick completeness proofs for some logics of conditionals. *Notre Dame J. Formal Logic*, 22:76–84, 1981.
[4] B. Chellas. *Modal Logic*. Cambridge University Press, 1980.
[5] G. D'Agostino and A. Visser. Finality regained: A coalgebraic study of Scott-sets and multisets. *Arch. Math. Logic*, 41:267–298, 2002.
[6] F. De Caro. Graded modalities II. *Stud. Log.*, 47:1–10, 1988.
[7] M. Fattorosi-Barnaba and C. Cerrato. Graded modalities III. *Stud. Log.*, 47:99–110, 1988.
[8] K. Fine. In so many possible worlds. *Notre Dame J. Formal Logic*, 13:516–520, 1972.
[9] N. Friedman and J. Y. Halpern. Plausibility measures and default reasoning. *J. ACM*, 48(4):648–685, 2001.
[10] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Logic for Programming and Automated Reasoning, LPAR 99*, vol. 1705 of *LNCS*, pp. 161–180. Springer, 1999.
[11] Y. Kazakov, U. Sattler, and E. Zolin. How many legs do I have? Non-simple roles in number restrictions revisited. In *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2007*, vol. 4790 of *LNCS*, pp. 303–317. Springer, 2007.
[12] A. Kurz and D. Pattinson. Coalgebraic modal logic of finite rank. *Math. Struct. Comput. Sci.*, 15:453–473, 2005.
[13] A. Kurz and J. Rosický. Strongly complete logics for coalgebras. Draft, July 2006.
[14] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inform. Comput.*, 94:1–28, 1991.
[15] D. Pattinson. Coalgebraic modal logic: Soundness, completeness and decidability of local consequence. *Theoret. Comput. Sci.*, 309:177–193, 2003.
[16] M. Pauly. A modal logic for coalitional power in games. *J. Logic Comput.*, 12:149–166, 2002.
[17] L. Schröder. A finite model construction for coalgebraic modal logic. *J. Log. Algebr. Prog.*, 73:97–110, 2007.
[18] L. Schröder and D. Pattinson. Rank-1 modal logics are coalgebraic. In *Theoretical Aspects of Computer Science, STACS 07*, vol. 4393 of *LNCS*, pp. 573–585. Springer, 2007. Full version to appear in *J. Log. Comput.*
[19] K. Segerberg. Notes on conditional logic. *Stud. Log.*, 48:157–168, 1989.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 685–696
www.stacs-conf.org

A STRONGER LP BOUND FOR FORMULA SIZE LOWER BOUNDS VIA CLIQUE CONSTRAINTS

KENYA UENO [1]

[1] Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo
E-mail address: kenya@is.s.u-tokyo.ac.jp

Abstract. We introduce a new technique proving formula size lower bounds based on the linear programming bound originally introduced by Karchmer, Kushilevitz and Nisan [11] and the theory of stable set polytope. We apply it to majority functions and prove their formula size lower bounds improved from the classical result of Khrapchenko [13]. Moreover, we introduce a notion of unbalanced recursive ternary majority functions motivated by a decomposition theory of monotone self-dual functions and give integrally matching upper and lower bounds of their formula size. We also show monotone formula size lower bounds of balanced recursive ternary majority functions improved from the quantum adversary bound of Laplante, Lee and Szegedy [15].

1. Introduction

Proving formula size lower bounds is a fundamental problem in complexity theory and also an extremely tough problem to resolve. A super-polynomial lower bound of a function in **NP** implies $\mathbf{NC}_1 \neq \mathbf{NP}$. There are a lot of techniques to prove formula size lower bounds, e.g. [7, 8, 11, 13, 14, 15, 16]. Laptente, Lee and Szegedy [15] introduced a technique based on the quantum adversary method [1] and gave a comparison with known techniques. In particular, they showed that their technique subsumes several known techniques such as Khrapchenko [13] and its extension [14]. The current best formula size lower bound is $n^{3-o(1)}$ by Håstad [7] and a key lemma used in the proof is also subsumed by the quantum adversary bound [15]. Karchmer, Kushilevitz and Nisan [11] introduced a technique proving formula size lower bounds called the linear programming (or LP) bound and showed that it cannot prove a lower bound larger than $4n^2$ for non-monotone formula size in general. Lee [16] proved that the LP bound [11] subsumes the quantum adversary bound [15] and Høyer, Lee and Špalek [8] introduced a stronger version of the quantum adversary bound.

Motivated by the result of Lee [16], we devise a stronger version of the LP bound by using an idea from the theory of stable set polytope, known as clique constraints [19]. Suggesting a stronger technique compared to the original LP bound [11] has possibilities to improve the best formula size lower bound because it subsumes many techniques including the key lemma of Håstad [7]. Moreover, our technique has various possibilities of extensions

1998 ACM Subject Classification: F.1.1 Models of Computation.
Key words and phrases: Computational and Structural Complexity.

STACS 2009
Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 685-696
http://drops.dagstuhl.de/opus/volltexte/2009/1808

such as rank constraints discussed in Section 6 and orthonormal constraints [6], each of which subsume clique constraints. Due to this extendability, it is difficult to show the limitation of our new technique.

To study the relative strength of our technique, we apply it to some families of Boolean functions. For each family, we have distinct motivation to investigate their formula size. Three kinds of Boolean functions treated in this paper are defined as follows. All of them are called monotone self-dual Boolean functions defined in the next section.

Definition 1.1. A majority function $\mathbf{MAJ}_{2l+1} : \{0,1\}^{2l+1} \mapsto \{0,1\}$ outputs 1 if the number of 1's in the input bits is greater than or equal to $l+1$ and 0 otherwise. We define unbalanced recursive ternary majority functions $\mathbf{URecMAJ}_3^h : \{0,1\}^{2h+1} \mapsto \{0,1\}$ as

$$\mathbf{URecMAJ}_3^h(x_1, \cdots, x_{2h+1}) = \mathbf{MAJ}_3(\mathbf{URecMAJ}_3^{h-1}(x_1, \cdots, x_{2h-1}), x_{2h}, x_{2h+1})$$

with $\mathbf{URecMAJ}_3^1 = \mathbf{MAJ}_3$. We also define balanced recursive ternary majority functions $\mathbf{BRecMAJ}_3^h : \{0,1\}^{3^h} \mapsto \{0,1\}$ as

$$\begin{aligned}\mathbf{BRecMAJ}_3^h(x_1, \cdots, x_{3^h}) = \mathbf{MAJ}_3(&\mathbf{BRecMAJ}_3^{h-1}(x_1, \cdots, x_{3^{h-1}}),\\ &\mathbf{BRecMAJ}_3^{h-1}(x_{3^{h-1}+1}, \cdots, x_{2\cdot 3^{h-1}}),\\ &\mathbf{BRecMAJ}_3^{h-1}(x_{2\cdot 3^{h-1}+1}, \cdots, x_{3^h}))\end{aligned}$$

with $\mathbf{BRecMAJ}_3^1 = \mathbf{MAJ}_3$. Through the paper, n means the number of input bits. Formula size and monotone formula size of a Boolean function f are denoted by $L(f)$ and $L_m(f)$, respectively.

Although our improvements of lower bounds seem to be slight, it breaks a stiff barrier (known as the certificate complexity barrier [15]) of previously known proof techniques. The best monotone upper and lower bounds of majority functions are $O(n^{5.3})$ [25] and $\lfloor n/2 \rfloor n$ [22], respectively. In the non-monotone case, the best formula size upper and lower bounds of majority functions are $O(n^{4.57})$ [20] and $\lceil n/2 \rceil^2$ $(= (l+1)^2$ when $n = 2l+1)$, respectively, which can be proven by the classical result of Khrapchenko [13]. In this paper, we slightly improve the non-monotone formula size lower bound while no previously known techniques has been able to improve it since 1971. In Section 4, we will prove $\frac{(l+1)^2}{1-\epsilon(l)} \leq L(\mathbf{MAJ}_{2l+1})$ where $\epsilon(l) = \frac{l^2(l+1)}{6\cdot\binom{2l+1}{l}}$. Here, $\binom{n}{k}$ denotes ${}_nC_k$. Since formula size takes an integral value, it implies a $(l+1)^2 + 1$ lower bound.

It is known that the class of monotone self-dual Boolean functions is closed under compositions (equivalently, in so-called Post's lattice [5, 21]). Any monotone self-dual Boolean functions can be decomposed into compositions of 3-bit majority functions [9]. A key observation for our proofs is that a communication matrix (defined in the next section) of a monotone self-dual Boolean function contains those of the 3-bit majority function as its submatrices. Ibaraki and Kameda [9] developed a decomposition theory of monotone self-dual Boolean functions in the context of mutual exclusions in distributed systems. The theory has been further investigated by [3, 4]. Given a monotone self-dual Boolean function f, we can decompose it as $f = \mathbf{MAJ}_3(x, f_1, (\mathbf{MAJ}_3(x, f_2, \mathbf{MAJ}_3(\cdots \mathbf{MAJ}_3(x, f_{k-1}, f_k)))))$ after decomposing $g = f(x = 0)$ into a conjunction of monotone self-dual functions $g = f_1 \wedge f_2 \wedge \cdots \wedge f_k$. It holds $\mathbf{URecMAJ}_3^h$ in its internal structure. To determine its formula size is of particular interest because it is related with efficiency of the decomposition scheme. In Section 5, we will prove $L(\mathbf{URecMAJ}_3^h) = L_m(\mathbf{URecMAJ}_3^h) = 4h + 1$.

Balanced recursive ternary majority functions have been studied in several contexts [10, 15, 17, 18, 23, 24], see [15] and [23] for details. Ambainis et al. [2] showed a quantum algorithm which evaluates a monotone formula of size N (or called AND-OR formula) in $N^{1/2+o(1)}$ time even if it is not balanced. This result implies $\mathbf{BRecMAJ}_3^h$ can be evaluated in $O(\sqrt{5}^h)$ time by the quantum algorithm because we have a formula size upper bound $L_m(\mathbf{BRecMAJ}_3^h) \leq 5^h$ as noted in [15]. Improving this result, Reichardt and Spalek [23] gave a quantum algorithm which evaluates $\mathbf{BRecMAJ}_3^h$ in $O(2^h)$ time. From this context, seeking the true bound of the monotone formula size of $\mathbf{BRecMAJ}_3^h$ is a very interesting research question. The quantum adversary bound [15] has a quite nice property written as $\mathbf{ADV}(f \cdot g) \geq \mathbf{ADV}(f) \cdot \mathbf{ADV}(g)$. It directly implies a formula size lower bound $4^h \leq L(\mathbf{BRecMAJ}_3^h)$. In Section 6, we will prove $20 \leq L_m(\mathbf{BRecMAJ}_3^2)$ and $4^h + \frac{13}{36} \cdot \left(\frac{8}{3}\right)^h \leq L_m(\mathbf{BRecMAJ}_3^h)$. This gives a slight improvement of the lower bound and means that the 4^h lower bound is at least not optimal in the monotone case.

2. Preliminaries

We define a total order $0 < 1$ between the two Boolean values. For Boolean vectors $\vec{x} = (x_1, \cdots, x_n)$ and $\vec{y} = (y_1, \cdots, y_n)$, we define $\vec{x} \leq \vec{y}$ if $x_i \leq y_i$ for all $i \in \{1, \cdots n\}$. A Boolean function f is called monotone if $\vec{x} \leq \vec{y}$ implies $f(\vec{x}) \leq f(\vec{y})$ for all $\vec{x}, \vec{y} \in \{0,1\}^n$. For a monotone Boolean function f, a Boolean vector $\vec{x} \in \{0,1\}^n$ is called minterm if $f(\vec{x}) = 1$ and $(\vec{y} \leq \vec{x}) \wedge (\vec{x} \neq \vec{y})$ implies $f(\vec{y}) = 0$ for any $\vec{y} \in \{0,1\}^n$ and called maxterm if $f(\vec{x}) = 0$ and $(\vec{x} \leq \vec{y}) \wedge (\vec{x} \neq \vec{y})$ implies $f(\vec{y}) = 1$ for any $\vec{y} \in \{0,1\}^n$. Sets of all minterms and maxterms of a monotone Boolean function f are denoted by $minT(f)$ and $maxT(f)$, respectively. A Boolean function f is called self-dual if $f(x_1, \cdots, x_n) = \overline{f(\overline{x}_1, \cdots, \overline{x}_n)}$ where $\overline{x}$ is the negation of x. Remark that, if a Boolean function f is self-dual, its communication matrix (see below) has some nice properties, e.g. $|X| = |Y|$.

A formula is a binary tree with leaves labeled by literals and internal nodes labeled by $\wedge$ and $\vee$. A literal is either a variable or the negation of a variable. A formula is called monotone if it does not have negations. It is known that all (monotone) Boolean functions can be represented by a (monotone) formula. The size of a formula is its number of leaves. We define the (monotone) formula size of a Boolean function f as the size of the smallest formula computing f.

Karchmer and Wigderson [12] characterize formula size of any Boolean function in terms of a communication game called the Karchmer-Wigderson game. In the game, given a Boolean function f, Alice gets an input $\vec{x}$ such that $f(\vec{x}) = 1$ and Bob gets an input $\vec{y}$ such that $f(\vec{y}) = 0$. The goal of the game is to find an index i such that $x_i \neq y_i$. They also characterize monotone formula size by a monotone version of the Karchmer-Wigderson game. In the monotone game, Alice gets a minterm $\vec{x}$ and Bob gets a maxterm $\vec{y}$. The goal of the monotone game is to find an index i such that $x_i = 1$ and $y_i = 0$. The number of leaves in a best communication protocol for the (monotone) Karchmer-Wigderson game is equal to the (monotone) formula size of f. From these characterizations, we consider communication matrices derived from the games.

Definition 2.1 (Communication Matrix)**.** Given a Boolean function f, we define its communication matrix as a matrix whose rows and columns are indexed by $X = f^{-1}(1)$ and $Y = f^{-1}(0)$, respectively. Each cell of the matrix contains indices i such that $x_i \neq y_i$. In

a monotone case, given a monotone Boolean function f, we define its monotone communication matrix as a matrix whose rows and columns are indexed by $X = minT(f)$ and $Y = maxT(f)$, respectively. Each cell of the matrix contains indices i such that $x_i = 1$ and $y_i = 0$. A combinatorial rectangle is a direct product $X' \times Y'$ where $X' \subseteq X$ and $Y' \subseteq Y$. A combinatorial rectangle $X' \times Y'$ is called monochromatic if every cell $(\vec{x}, \vec{y}) \in X' \times Y'$ contains the same index i. We call a cell singleton if it contains just one index.

The minimum number of disjoint monochromatic rectangles which exactly cover all cells in the (monotone) communication matrix gives a lower bound for the number of leaves of a best communication protocol for the (monotone) Karchmer-Wigderson game. Thus, we obtain the following bound.

Theorem 2.2 (Rectangle Bound [12])**.** *The minimum size of an exact cover by disjoint monochromatic rectangles for the communication matrix (or monotone communication matrix) associated with a Boolean function f gives a lower bound of $L(f)$ (or $L_m(f)$).*

3. A Stronger Linear Programming Bound via Clique Constraints

In this study, we devise a new technique proving formula size lower bounds based on the LP bound [11] with clique constraints. We assume that readers are familiar with the basics of the linear and integer programming theory. Karchmer, Kushilevitz and Nisan [11] formulate the rectangle bound as an integer programming problem and give its LP relaxation. Given a (monotone) communication matrix, it can be written as $\min \sum_r x_r$ such that $\sum_{r \ni c} x_r = 1$ for each cell c in the matrix and $x_r \geq 0$ for each monochromatic rectangle r. The dual problem can be written as $\max \sum_c w_c$ such that $\sum_{c \in r} w_c \leq 1$ for each monochromatic rectangle r. Here, each variable w_c is indexed by a cell c in the matrix. From the duality theorem, showing a feasible solution of the dual problem gives a formula size lower bound.

Now, we introduce our stronger LP bound using clique constraints from the theory of stable set polytope. We assume that each monochromatic rectangle is a node of a graph. We connect two nodes by an edge if the two corresponding monochromatic rectangles intersect. If a set of monochromatic rectangles q compose a clique in the graph, we add a constraint $\sum_{r \in q} x_r \leq 1$ to the primal problem of the LP relaxation. This constraint is valid for all integral solutions since we consider the disjoint cover problem. That is, we can assign the value 1 to at most 1 rectangle in a clique for all integral solutions under the condition of disjointness. The dual problem can be written as $\max \sum_c w_c + \sum_q z_q$ such that $\sum_{c \in r} w_c + \sum_{q \ni r} z_q \leq 1$ for each monochromatic rectangle r and $z_q \leq 0$ for each clique q. Intuitively, this formulation can be interpreted as follows. Each cell c is assigned a weight w_c. The summation of weights over all cells in a monochromatic rectangle is limited to 1. This limit is relaxed by 1 if it is contained by a clique. Thus, the limit of the total weight for a monochromatic rectangle contained by k distinct cliques is $k + 1$.

By using clique constraints, we obtain the following matching lower bound for the formula size of the 3-bit majority function while the original LP bound cannot prove a lower bound larger than 4.5. In our proofs, we utilize the following property of combinatorial rectangles which is trivial from the definition. If a rectangle contains two cells (α_1, β_1) and (α_2, β_2), it also contains both (α_1, β_2) and (α_2, β_1). A notion of singleton cells also occupies an important role for our proofs because there are no monochromatic rectangles which contain different kinds of singleton cells.

Theorem 3.1. $L(\mathbf{MAJ}_3) = L_m(\mathbf{MAJ}_3) = 5$

Proof. We have a monotone formula $(x_1 \wedge x_2) \vee ((x_1 \vee x_2) \wedge x_3)$ for $\mathbf{MAJ}_3$. From the definition, $L(\mathbf{MAJ}_3) \leq L_m(\mathbf{MAJ}_3)$. To prove $L(\mathbf{MAJ}_3) \geq 5$, we consider a communication matrix of the 3-bit majority function whose rows and columns are restricted to minterms and maxterms, respectively.

	100	010	001
110	2	1	1,2,3
101	3	1,2,3	1
011	1,2,3	3	2

Figure 1: The Communication Matrix of $\mathbf{MAJ_3}$

In the dual problem, we assign weights 1 for all singleton cells and 0 for other cells. There are 6 singleton cells and hence the total weight is 6. We take a clique q composed of monochromatic rectangles containing two singleton cells. It is clear that every pair of monochromatic rectangles contained by q intersect at some cell. We assign $z_q = -1$. Then, the objective function of the dual problem becomes $5 = 6 - 1$.

Now, we show that all constraints of the dual problem are satisfied. First, we consider a monochromatic rectangle which contains at most one singleton cell. In this case, the constraint is clearly satisfied because the summation of weights in the monochromatic rectangle is less than or equal to 1. Then, we consider a monochromatic rectangle which contains two singleton cells. In this case, the summation of weights in the monochromatic rectangle is 2. However, it is contained by the clique q. It implies that the limit of the total weight is relaxed by 1. Thus, the constraint is satisfied. There are no monochromatic rectangles which contain more than 3 singleton cells because a rectangle which contains more than two kinds of singleton cells is not monochromatic. ∎

4. Formula Size of Majority Functions

In this section, we show a non-monotone formula size lower bound of majority functions improved from the classical result of Khrapchenko [13].

Theorem 4.1. $L(\mathbf{MAJ}_{2l+1}) \geq \dfrac{(l+1)^2}{1-\epsilon(l)}$ *where* $\epsilon(l) = \dfrac{l^2(l+1)}{6 \cdot \binom{2l+1}{l}}$.

Proof. We consider a communication matrix of a majority function with $2l+1$ input bits whose rows and columns are restricted to minterms and maxterms, respectively. Let $m = \binom{2l+1}{l}$, which is equal to both the number of rows and columns. Then, the number of all cells is m^2. The number of singleton cells is $(l+1)m$ and hence the number of singleton cells for each index is $\frac{(l+1)m}{2l+1}$. The number of cells with 3 indices is $\binom{l+1}{2} \cdot l \cdot m = \frac{l^2(l+1)m}{2}$ because we can obtain a maxterm by flipping two bits of 1's to 0's and one bit of 0 to 1 for each minterm.

We consider 3×3 submatrices in the following way. From $2l+1$ input bits, we fix arbitrary $2l-2$ bits and assume that they have the same number of 0's and 1's. Then, we consider the remaining 3 bits. If the $2l+1$ input bits compose a minterm, the 3

bits are 110 or 101 or 011. If the $2l+1$ input bits compose a maxterm, the 3 bits are 100 or 010 or 001. Thus, we have a 3×3 submatrix, which has the same structure as the communication matrix of the 3-bit majority function as Figure 1. The number of submatrices is $\binom{2l+1}{3} \cdot \binom{2l-2}{l-1} = \frac{l^2(l+1)m}{6}$. Each submatrix has 6 singleton cells and 3 cells each of which has 3 indices corresponding to the remaining 3 bits. Note that each cell with 3 indices in any submatrix is not contained by other submatrices. In other words, all the $\frac{l^2(l+1)m}{2}$ cells with 3 indices are exactly partitioned into the $\frac{l^2(l+1)m}{6}$ submatrices.

We assign weights a for all singleton cells, 0 for cells with 3 indices and b for other cells, which have more than 3 indices. Note that there are no cells with 2 indices. We consider $\frac{l^2(l+1)m}{6}$ clique constraints assigned weights c (≤ 0) for all the $\frac{l^2(l+1)m}{6}$ submatrix. That is, we have a clique constraint for each submatrix similar to the proof of Theorem 3.1. More precisely, a clique associated with a submatrix is composed of monochromatic rectangles which contain two singleton cells in the submatrix.

Then, the objective function of the dual problem is written as

$$\max_{a,b,c}(l+1)m \cdot a + \left(m^2 - (l+1)m - \frac{l^2(l+1)m}{2}\right) \cdot b + \frac{l^2(l+1)m}{6} \cdot c. \tag{4.1}$$

Now, we fix $c = 2b \leq 0$. Then, we have

$$\max_{a,b}(l+1)m \cdot a + \left(m^2 - (l+1)m - \frac{l^2(l+1)m}{6}\right) \cdot b. \tag{4.2}$$

We assume that a monochromatic rectangle contains k singleton cells and consider all possible pairs of 2 singleton cells taken from the k singleton cells. If a pair is in the same submatrix, the monochromatic rectangle is contained by a clique associated with the submatrix. If a pair is not in the same submatrix, the monochromatic rectangle contains two cells which are assigned weights b because they have more than 3 indices. Thus, if the following inequality is satisfied

$$k \cdot a + (k^2 - k) \cdot b \leq 1 \tag{4.3}$$

for any integer k ($1 \leq k \leq \frac{(l+1)m}{2l+1}$), all constraints of the dual problem are satisfied when $c = 2b$.

We can maximize (4.2) by assuming that the inequality is saturated when $k = \frac{m}{l+1} - \frac{l^2}{6}$ as it satisfies $\frac{k^2-k}{k} = \frac{m^2-(l+1)m-\frac{l^2(l+1)m}{6}}{(l+1)m}$. In this case, we have (4.2) $= \frac{(l+1)m}{\frac{m}{l+1}-\frac{l^2}{6}} = \frac{(l+1)^2m}{m-\frac{1}{6}l^2(l+1)}$ and obtain the lower bound. ∎

5. Formula Size of Unbalanced Recursive Ternary Majority Functions

In this section, we show the following matching bound of formula size for unbalanced recursive ternary majority functions.

Theorem 5.1. $L(\mathbf{URecMAJ}_3^h) = L_m(\mathbf{URecMAJ}_3^h) = 4h+1$

Proof. First, we look at the monotone formula size upper bound. Recall that a monotone formula of the 3-bit majority function can be written as $(x_1 \wedge x_2) \vee ((x_1 \vee x_2) \wedge x_3)$. The important point here is that the literal x_3 appears only once. We construct $(x_{2h} \wedge x_{2h+1}) \vee ((x_{2h} \vee x_{2h+1}) \wedge x_{2h-1})$ and replace x_{2h-1} by a monotone formula representing $\mathbf{URecMAJ}_3^{h-1}$. A recursive construction yields a $4h+1$ monotone formula for $\mathbf{URecMAJ}_3^h$.

Then, we show the non-monotone formula size lower bound. Before using clique constraints, we consider the original LP bound. We restrict the communication matrix of $\mathbf{URecMAJ}_3^h$ to a submatrix S_h whose rows and columns are minterms and maxterms, respectively. We can interpret it in the following recursive way as Figure 2.

	00	10	01
11	$2h, 2h+1$	$2h+1$	$2h$
01	$2h+1$	T_{h-1}	S_{h-1}
10	$2h$	S_{h-1}	T_{h-1}

Figure 2: Recursive Structure of S_h for $\mathbf{URecMAJ}_3^h$ ($h \geq 2$)

In the figure, "11" denotes a minterm which has 1 in the $2h$-th and $(2h+1)$-th bits and 0 in other $(2h-1)$ bits. Minterms denoted by "01" has 0 in the $2h$-th bit and 1 in the $(2h+1)$-th bit and other $(2h-1)$ bits of them are determined by a recursive way from minterms of $\mathbf{URecMAJ}_3^{h-1}$. Minterms denoted by "10" has 1 in the $2h$-th bit and 0 in the $(2h+1)$-th bit and other $(2h-1)$ bits of them are also determined by the recursive way. "00", "10" and "01" denote maxterms which are similarly defined as minterms. A submatrix T_{h-1} does not contain singleton cells because all cells in T_{h-1} contains indices $\{2h, 2h+1\}$ with indices of corresponding cell in S_{h-1}. S_h contains two S_{h-1}. Thus, the number of singleton cells duplicate in each recursion.

We consider the minimum submatrix $\mathbf{ALL}{-}\mathbf{S}_1$ in S_h which contains all three kinds of singleton cells $\{1\}$, $\{2\}$ and $\{3\}$. Note that $\mathbf{ALL}{-}\mathbf{S}_1$ does not contain any other kinds of singleton cells because it only contains cells in S_1 and T_l ($2 \leq l \leq h-1$). A submatrix S_1 is equivalent to a communication matrix of the 3-bit majority function. The total number of singleton cells $\{1\}$, $\{2\}$ and $\{3\}$ is $3 \cdot 2^h$. Both the number of rows and columns of $\mathbf{ALL}{-}\mathbf{S}_1$ is equal to $3 \cdot 2^{h-1}$ because S_1's duplicate $(h-1)$-times and does not have any common rows and columns. Hence, the number of all cells in $\mathbf{ALL}{-}\mathbf{S}_1$ is $9 \cdot 4^{h-1}$. We assign weights a for all singleton cells in $\mathbf{ALL}{-}\mathbf{S}_1$ and weights b for all other cells in $\mathbf{ALL}{-}\mathbf{S}_1$. Then, the total weight of all cells in $\mathbf{ALL}{-}\mathbf{S}_1$ is written as follows:

$$\max_{a,b} 3 \cdot 2^h \cdot a + \left(9 \cdot 4^{h-1} - 3 \cdot 2^h\right) \cdot b. \tag{5.1}$$

We consider constraints of the dual problem as $k \cdot a + (k^2 - k) \cdot b \leq 1$ for all integer k ($1 \leq k \leq 2^h$). We assume this inequality is saturated if and only if $k = 3 \cdot 2^{h-2}$. Then, we get $a = \frac{24 \cdot 2^h - 16}{9 \cdot 4^h}$ and $b = -\frac{16}{9 \cdot 4^h}$. In this case, (5.1) $= 4$.

Next, we consider singleton cells $\{2l\}$ and $\{2l+1\}$ ($2 \leq l \leq h$). We partition singleton cells $\{2l\}$ into two sets named vertical cells X_{2l} and horizontal cells Y_{2l} which are in (10,00) and (11,01) of each S_l in S_h, respectively. Similarly, we partition singleton cells $\{2l+1\}$ into two sets named vertical cells X_{2l+1} and horizontal cells Y_{2l+1} which are in (01,00) and (11,10) of each S_l in S_h, respectively. We restrict these sets to the minimum subsets $X'_{2l} \subset X_{2l}$, $X'_{2l+1} \subset X_{2l+1}$, $Y'_{2l} \subset Y_{2l}$ and $Y'_{2l+1} \subset Y_{2l+1}$ so as to satisfy the following condition: If a monochromatic rectangle contains all cells in $X'_{2l} \cup X'_{2l+1} \cup Y'_{2l} \cup Y'_{2l+1}$, it also contains all cells in $\mathbf{ALL}{-}\mathbf{S}_1$. Note that rows and columns of singleton cells $\{2l\}$ and $\{2l+1\}$ dominate those of singleton cells $\{1\}$, $\{2\}$ and $\{3\}$. So, we have $|X'_{2l}| = |X'_{2l+1}| = |Y'_{2l}| = |Y'_{2l+1}| = 3 \cdot 2^{h-2}$. We assign weights $\frac{1}{3 \cdot 2^{h-2}}$ for all singleton cells in $X'_{2l} \cup X'_{2l+1} \cup Y'_{2l} \cup Y'_{2l+1}$ and 0 for other

cells at (11,00) of each S_l and cells outside $\mathbf{ALL}-\mathbf{S}_1$. A monochromatic rectangle which contains x cells in X'_{2l} and y in from Y'_{2l} also contains $x \cdot y$ cells in $\mathbf{ALL}-\mathbf{S}_1$ which are assigned weights b. The same thing is true for the case of X'_{2l+1} and Y'_{2l+1}. Because we have

$$(x+y) \cdot \frac{4}{3 \cdot 2^h} - xy \cdot \frac{16}{9 \cdot 4^h} \leq 1 \tag{5.2}$$

for all $0 \leq x, y \leq 3 \cdot 2^{h-2}$, all constraints of the dual problem are satisfied. The total weight of singleton cells $\{2l\}$ and $\{2l+1\}$ is 4. So, the total weight of all cells in S_h now becomes $4h$.

Now, we incorporate clique constraints. The number of S_1 in S_h is 2^{h-1}. We change weights of all non-singleton cells in submatrices S_1 from b to 0. On behalf of them, we add a clique constraint for each S_1 in S_h. Then, (5.1) becomes

$$\max_{a,b,c} 3 \cdot 2^h \cdot a + \left(9 \cdot 4^{h-1} - 3 \cdot 2^h - 3 \cdot 2^{h-1}\right) \cdot b + 2^{h-1} \cdot c. \tag{5.3}$$

where c is a weight assigned for each clique constraint. If we take $a = \frac{24 \cdot 2^h - 16}{9 \cdot 4^h}$, $b = -\frac{16}{9 \cdot 4^h}$ and $c = 2b$, all constraints of the dual problem are satisfied and (5.3) $= 4 + \frac{8}{9} \cdot 2^{-h}$. Consequently, the total weight is $4h + \frac{8}{9} \cdot 2^{-h}$. Since formula size must be an integer, we have shown the theorem. ∎

6. Monotone Formula Size of Balanced Recursive Ternary Majority Functions

In this section, we show monotone formula size lower bounds of balanced recursive ternary majority functions. For this purpose, we consider rank constraints, which are generalizations of clique constraints. Similarly to the case of clique constraints, we consider a graph composed of monochromatic rectangles and its induced subgraph g. We consider a constraint $\sum_{r \in g} x_r \leq \alpha(g)$ where $\alpha(g)$ is the stability number of g. This constraint is valid because we can assign 1 at most $\alpha(g)$ rectangles in g for any integral solution. The dual problem can be written as $\max \sum_c w_c + \sum_q z_q + \sum_g \alpha(g) z_g$ such that $\sum_{c \in r} w_c + \sum_{q \ni r} z_q + \sum_{g \ni r} z_g \leq 1$ for each monochromatic rectangle r, $z_q \leq 0$ for each clique q and $z_g \leq 0$ for each subgraph g.

First, we consider the case of height 2. By using clique constraints and rank constraints, we prove the following improved monotone formula size lower bound while we know that the original LP bound cannot prove a lower bound larger than 16.5.

Theorem 6.1. $L_m(\mathbf{BRecMAJ}_3^2) \geq 20$

Proof. There are 27 minterms and 27 maxterms for the recursive ternary majority function of height 2. Among them, we choose the following 9 minterms

110,110,000	101,101,000	011,011,000
110,000,110	101,000,101	011,000,011
000,110,110	000,101,101	000,011,011

and 9 maxterms

111,100,100	111,010,010	111,001,001
100,111,100	010,111,010	001,111,001
100,100,111	010,010,111	001,001,111.

From these 9 minterms and 9 maxterms, a submatrix of the communication matrix can be described as Figure 3. In the figure, we abbreviate a minterm e.g. 101,101,000 by 110 and 101, which represent the second level and the first level structure of the 9 bits, respectively. Notice that all minterms which we choose have the same structure in all 3-bits minterm blocks at the first level. The same thing is true for all 9 maxterms.

		100			010			001		
		100	010	001	100	010	001	100	010	001
110	110	5	4	4,5	2	1	1,2	2,5	1,4	1,2,4,5
	101	6	4,6	4	3	1,3	1	3,6	1,3,4,6	1,4
	011	5,6	6	5	2,3	3	2	2,3,5,6	3,6	2,5
101	110	8	7	7,8	2,8	1,7	1,2,7,8	2	1	1,2
	101	9	7,9	7	3,9	1,3,7,9	1,7	3	1,3	1
	011	8,9	9	8	2,3,8,9	3,9	2,8	2,3	3	2
011	110	5,8	4,7	4,5,7,8	8	7	7,8	5	4	4,5
	101	6,9	4,6,7,9	4,7	9	7,9	7	6	4,6	4
	011	5,6,8,9	6,9	5,8	8,9	9	8	5,6	6	5

Figure 3: A Submatrix of the Communication Matrix for $\mathbf{BRecMAJ}_3^2$

		100			010			001		
		100	010	001	100	010	001	100	010	001
110	110	1	2	3	4	5	6	7	8	9
	101	10	11	12	13	14	15	16	17	18
	011	19	20	21	22	23	24	25	26	27
101	110	28	29	30	31	32	33	34	35	36
	101	37	38	39	40	41	42	43	44	45
	011	46	47	48	49	50	51	52	53	54
011	110	55	56	57	58	59	60	61	62	63
	101	64	65	66	67	68	69	70	71	72
	011	73	74	75	76	77	78	79	80	81

Figure 4: Serial Numbers for 81 cells of the Submatrix

To describe 12 cliques $q_1, \cdots, q_{12}$ and a induced subgraph g whose stability number is 4, we give serial numbers for 81 cells as Figure 4. We take the following 12 cliques each of which consists of 3 pairs of 2 singleton cells:

{ (5, 15), (4, 24), (13, 23) }, { (35, 45), (34, 54), (43, 53) },
{ (2, 12), (1, 21), (10, 20) }, { (62, 72), (61, 81), (70, 80) },
{ (29, 39), (28, 48), (37, 47) }, { (59, 69), (58, 78), (67, 77) },
{ (5, 35), (2, 62), (29, 59) }, { (15, 45), (12, 72), (39, 69) },
{ (4, 34), (1, 61), (28, 58) }, { (24, 54), (21, 81) (48, 78) },
{ (13, 43), (10, 70), (37, 67) }, { (23, 53), (20, 80), (47, 77) }.

For each combination of 3 pairs, it is easy to verify that rectangles each of which contains both of two singleton cells from one of the 3 pairs compose a clique.

Next, we consider the following 18 pairs of singleton cells which induce the subgraph g:

(5, 45), (15, 35), (4, 54), (24, 34), (13, 53), (23, 43), (2, 72), (12, 62), (1, 81),
(21, 61), (10, 80), (20, 70), (29, 69), (39, 59), (28, 78), (48, 58), (37, 77), (47, 67).

If a rectangle contain both of two singleton cells from one of 18 pairs, it also contains 2 cells from 9 cells { 9, 17, 25, 33, 41, 49, 57, 65, 73 }. Thus, we can choose at most 4 pairs without conflicts from 18 pairs. It implies that the stability number of g is 4.

Notice that all these 12 cliques and the subgraph cover all pairs of two singleton cells which have the same index. We assign 1 for all 36 singleton cells in this submatrix and 0 for other cells. We take $z_{q_1} = \cdots = z_{q_{12}} = z_g = -1$. Then, the objective value of the dual problem becomes $36 - 12 - 4 = 20$. If a rectangle contains at most one singleton cell, the constraint of the dual problem is trivially satisfied. If a rectangle contains k $(2 \leq k \leq 4)$ singleton cells, it is covered by $k-1$ cliques or $k-2$ cliques plus the subgraph g. So, the constraint is also satisfied. As a consequence, we obtain the formula size lower bound. ∎

Note that we need a much more complicated argument to look at the non-monotone case, which we do not investigate in this paper, because singleton cells in the monotone communication matrix are not singleton in the non-monotone communication matrix.

In the general monotone case, we can prove a slightly better lower bound than the quantum adversary bound [15], which shows a 4^h lower bound.

Theorem 6.2. $L_m(\mathbf{BRecMAJ}_3^h) \geq 4^h + \frac{13}{36} \cdot \left(\frac{8}{3}\right)^h$ $(h \geq 2)$

Proof. First, we choose 3^h minterms and 3^h maxterms from 3^h input bits of $\mathbf{BRecMAJ}_3^h$ so as to have the same structure in the 1st, 2nd, $\cdots$ and h-th levels in the following sense. In the l-th level, we have 3^{h-l} bits which are recursively constructed from lower levels in the following way. We partition 3^l bits into 3^{l-1} blocks each of which contains consecutive 3 bits. For each block of 3 bits, we replace them into 1 bit which is the output of $\mathbf{MAJ}_3$ with the 3 bits. Then, we get $3^{h-(l+1)}$ bits. We have 3^h bits as input bits in the first level and can construct them for each level by induction. If all of 3^{l-1} blocks have the same 3 bits except 000 and 111 in the case of minterms and maxterms, respectively, we call that they have the same structure in the l-the level. There are 3^h minterms and 3^h maxterms because we have 3 choices in each level. We consider the submatrix whose rows and columns are composed of these 3^h minterms and 3^h maxterms, respectively.

From another viewpoint, we can interpret it as a recursively construction of the submatrix S_h of the communication matrix of $\mathbf{BRecMAJ}_3^h$ as follows. We define $S_h(k)$ $(k = 1, 2, 3)$ as a matrix such that some cell of $S_h(k)$ contains an index $(k-1) \cdot 3^h + i$ if and only if the corresponding cell of S_h contains an index i. By induction, we can see that the number of all cells and singleton cells in S_h is 9^h and 6^h, respectively. Singleton cells of each index from 3^h bits in S_h is 2^h. Indices of cells in $T_h(1,2)$, $T_h(2,3)$ and $T_h(2,3)$ in Figure 5 can be determined from the property of combinatorial rectangles, but we do not go to the details because we will assign the same weight for all these cells in each level.

Before using clique and rank constraints, we consider the original LP bound. We assign weights a for all singleton cells, b for other cells in the submatrix and 0 for all cells in the outside of the submatrix. Then, the objective value of the dual problem is written as

$$\max_{a,b} 6^h \cdot a + (9^h - 6^h) \cdot b. \tag{6.1}$$

	100	010	001
110	$S_{h-1}(2)$	$S_{h-1}(1)$	$T_{h-1}(1,2)$
101	$S_{h-1}(3)$	$T_{h-1}(2,3)$	$S_{h-1}(1)$
011	$T_{h-1}(2,3)$	$S_{h-1}(3)$	$S_{h-1}(2)$

Figure 5: Recursive Structure of S_h for $\mathbf{BRecMAJ}_3^h$ ($h \geq 2$)

If a rectangle contains k singleton cells, it also contains at least $k^2 - k$ cells which are not singleton. Thus, if $k \cdot a + (k^2 - k) \cdot b \leq 1$ is satisfied for all integer k ($1 \leq k \leq 2^h$), then all constraints of the dual problem are also satisfied. We assume that the inequality is saturated if and only if $k = (3/2)^h$. Then, we get $a = \frac{2 \cdot 6^h - 4^h}{9^h}$ and $b = -\frac{4^h}{9^h}$. In this case, we have $(6.1) = 4^h$.

Now, we incorporate clique and rank constraints. We change weights of all cells except singleton cells in all S_2's in the second level from b to 0. Then, we add 12 clique constraints and a rank constraint for each S_2 in the second level by following the way of Theorem 6.1. Let c and d be values assigned for every clique and rank constraints, respectively. Then, the objective value of the dual problem is

$$\max_{a,b,c,d} 6^h \cdot a + (9^h - 81 \cdot 6^{h-2}) \cdot b + 12 \cdot 6^{h-2} \cdot c + 4 \cdot 6^{h-2} \cdot d. \tag{6.2}$$

If we take $c = d = 2b$, then we have $(6.2) = 6^h \cdot a + (9^h - 49 \cdot 6^{h-2}) \cdot b = 4^h + \frac{13}{36} \cdot \left(\frac{8}{3}\right)^h$. Since all weights which are changed from b to 0 are exactly compensated by clique and rank constraints, all constraints of the dual problem are satisfied. ∎

We do not exhaust the potential of our new method and have possibilities to improve the lower bound. For example, we can improve the lower bound as $4^h + c \cdot \left(\frac{8}{3}\right)^h$ for some constant c by further detailed analysis in constantly higher levels.

7. Conclusions

In this paper, we devised the new technique proving formula size lower bounds and showed improved formula size lower bounds of some families of monotone self-dual Boolean functions such as majority functions, unbalanced and balanced recursive ternary majority functions. We hope that our method will be able to improve formula size lower bounds for any monotone self-dual Boolean function and even much broader classes of Boolean functions. Whether our technique (or its extensions) can break the $4n^2$ barrier and improve the best formula size lower bound remains open.

Acknowledgment

The author is grateful to Norie Fu for cooperating computational experiments, Troy Lee for suggesting to look at compositions of Boolean functions, Patric Östergård for useful information about the exact cover problem, and Kazuhisa Makino for helpful discussion. This research is supported by Research Fellowship for Young Scientists from Japan Society for the Promotion of Science (JSPS) and Grant-in-Aid for JSPS Fellows.

References

[1] A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002.

[2] A. Ambainis, A. M. Childs, B. Reichardt, R. Špalek, and S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 363–372, 2007.

[3] J. C. Bioch and T. Ibaraki. Decompositions of positive self-dual boolean functions. *Discrete Mathematics*, 140(1-3):23–46, 1995.

[4] J. C. Bioch, T. Ibaraki, and K. Makino. Minimum self-dual decompositions of positive dual-minor boolean functions. *Discrete Applied Mathematics*, 96-97:307–326, 1999.

[5] E. Böhler, N. Creignou, S. Reith, and H. Vollmer. Playing with boolean blocks, part I: Post's lattice with applications to complexity theory. *ACM SIGACT News*, 34(4):38–52, 2003.

[6] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[7] J. Håstad. The shrinkage exponent of De Morgan formulas is 2. *SIAM Journal on Computing*, 27(1):48–64, Feb. 1998.

[8] P. Høyer, T. Lee, and R. Špalek. Negative weights make adversaries stronger. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 526–535, 2007.

[9] T. Ibaraki and T. Kameda. A theory of coteries: Mutual exclusion in distributed systems. *IEEE Transactions on Parallel and Distributed Computing*, PDS-4(7):779–794, July 1993.

[10] T. S. Jayram, R. Kumar, and D. Sivakumar. Two applications of information complexity. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 673–682, 2003.

[11] M. Karchmer, E. Kushilevitz, and N. Nisan. Fractional covers and communication complexity. *SIAM Journal on Discrete Mathematics*, 8(1):76–92, Feb. 1995.

[12] M. Karchmer and A. Wigderson. Monotone circuits for connectivity require super-logarithmic depth. *SIAM Journal on Discrete Mathematics*, 3(2):255–265, May 1990.

[13] V. M. Khrapchenko. Complexity of the realization of a linear function in the case of π-circuits. *Mathematical Notes*, 9:21–23, 1971.

[14] E. Koutsoupias. Improvements on Khrapchenko's theorem. *Theoretical Computer Science*, 116(2):399–403, Aug. 1993.

[15] S. Laplante, T. Lee, and M. Szegedy. The quantum adversary method and classical formula size lower bounds. *Computational Complexity*, 15(2):163–196, 2006.

[16] T. Lee. A new rank technique for formula size lower bounds. In *Proceedings of the 24th Annual Symposium on Theoretical Aspects of Computer Science*, volume 4393 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2007.

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

[18] R. O'Donnell. Hardness amplification within NP. *Journal of Computer and System Sciences*, 69(1):68–94, 2004.

[19] M. Padberg. On the facial structure of the set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[20] M. S. Paterson, N. Pippenger, and U. Zwick. Optimal carry save networks. In *Boolean function complexity*, volume 169 of *London Mathematical Society Lecture Note Series*, pages 174–201. Cambridge University Press, 1992.

[21] E. L. Post. *The two-valued iterative systems of mathematical logic*, volume 5 of *Annals Mathematical Studies*. Princeton University Press, 1941.

[22] J. Radhakrishnan. Better lower bounds for monotone threshold formulas. *Journal of Computer and System Sciences*, 54(2):221–226, Apr. 1997.

[23] B. Reichardt and R. Spalek. Span-program-based quantum algorithm for evaluating formulas. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 103–112, 2008.

[24] M. E. Saks and A. Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 29–38, 1986.

[25] L. G. Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5(3):363–366, Sept. 1984.

Symposium on Theoretical Aspects of Computer Science 2009 (Freiburg), pp. 697–708
www.stacs-conf.org

EXTRACTING THE KOLMOGOROV COMPLEXITY OF STRINGS AND SEQUENCES FROM SOURCES WITH LIMITED INDEPENDENCE

MARIUS ZIMAND [1]

[1] Department of Computer and Information Sciences
Towson University
E-mail address: mzimand@towson.edu
URL: http://triton.towson.edu/~mzimand

Abstract. An infinite binary sequence has randomness rate at least σ if, for almost every n, the Kolmogorov complexity of its prefix of length n is at least σn. It is known that for every rational $\sigma \in (0,1)$, on one hand, there exists sequences with randomness rate σ that can not be effectively transformed into a sequence with randomness rate higher than σ and, on the other hand, any two independent sequences with randomness rate σ can be transformed into a sequence with randomness rate higher than σ. We show that the latter result holds even if the two input sequences have linear dependency (which, informally speaking, means that all prefixes of length n of the two sequences have in common a constant fraction of their information). The similar problem is studied for finite strings. It is shown that from any two strings with sufficiently large Kolmogorov complexity and sufficiently small dependence, one can effectively construct a string that is random even conditioned by any one of the input strings.

1. Introduction

The randomness rate of an object is the ratio between the information in the object and its length. An informal principle states that no reasonable transformation can guarantee an increase of the randomness rate. The principle has different instantiations depending on the meaning of "object", "information," and "reasonable transformation." For example, if f is a mapping of the set of n-bit strings to the set of m-bit strings, then there is a distribution X on the set of n-bit strings with Shannon entropy $n/2$ (i.e., the randomness rate of X is $1/2$) and the Shannon entropy of $f(X)$ is $\leq m/2$ (i.e., the randomness rate of $f(X)$ is $\leq 1/2$). Thus no transformation f guarantees that its output has a randomness rate higher than that of its input. The case of infinite binary sequences (in short, sequences) is very interesting and has been recently the subject of intensive research. We say that a sequence x has randomness rate at least σ if $K(x\upharpoonright n) \geq \sigma \cdot n$, for all sufficiently large n. Here, $x\upharpoonright n$ is the prefix of length n of x and $K(\cdot)$ is the Kolmogorov complexity. A related notion is that of effective Hausdorff dimension of a sequence x, defined as: $\dim(\mathrm{x}) = \liminf \mathrm{K}(\mathrm{x}\upharpoonright\mathrm{n})/\mathrm{n}$. Reiman and

Key words and phrases: algorithmic information theory, computational complexity, Kolmogorov complexity, randomness extractors.

The author is supported by NSF grant CCF 0634830.

Terwijn [Rei04] have asked whether for any sequence x with $\dim(\mathrm{x}) = 1/2$ there exists an effective transformation (formally, a Turing reduction) f such that $\dim(\mathrm{f(x)}) > 1/2$. Initially, some partial negative results have been obtained for transformations f with certain restrictions. Reimann and Terwijn [Rei04, Th 3.10] have shown that the answer is NO if we require that f is a many-one reduction. This result has been extended by Nies and Reimann [NR06] to wtt-reductions. Bienvenu, Doty, and Stephan [BDS07] have obtained an impossibility result for the general case of Turing reductions, which, however, is valid only for *uniform* transformations. More precisely, building on the result of Nies and Reimann, they have shown that for all constants c_1 and c_2, with $0 < c_1 < c_2 < 1$, there is no Turing reduction f such that for any sequence x with $\dim(\mathrm{x}) \geq \mathrm{c}_1$ has the property that $\dim(\mathrm{f(x)}) \geq \mathrm{c}_2$. In other words, loosely speaking, no effective uniform transformation is able to raise the randomness rate from c_1 to c_2. Finally, very recently, Miller [Mil08] has fully solved the original question, by constructing a sequence x with $\dim(\mathrm{x}) = 1/2$ such that, for any Turing reduction f, $\dim(\mathrm{f(x)}) \leq 1/2$ (or $f(x)$ does not exist).

On the other hand, Zimand [Zim08] has shown that it is possible to increase the randomness rate if the input consists of *two sequences* that enjoy a certain type of *independence*. Namely, we say that two sequences x and y are finitary-independent[1] if for all n and m,

$$K(x\restriction n\ y\restriction m) \geq K(x\restriction n) + K(y\restriction m) - O(\max(\log n, \log m)). \tag{1.1}$$

In [Zim08], it is shown that for any constant $0 < \tau \leq 1$, there is a Turing reduction f such that, for any finitary-independent sequences x and y, both with randomness rate $\geq \tau$, it holds that $f(x, y)$ has randomness rate arbitrarily close to 1 (in particular, $\dim(\mathrm{f(x, y)}) = 1$). Moreover f is a truth-table reduction and also f is uniform in τ.

To summarize, if we start with one source, it is impossible to effectively increase the randomness rate, while if we start with two finitary-independent sequences it is possible to increase the randomness rate to close to 1 in a uniform and truth-table manner.

It is clear that the independence requirement plays an important role in the positive result. Since independence can be quantified, it is interesting to see what level of independence is needed for a positive result.

For a function $d : \mathbb{N} \to \mathbb{R}$, we say that strings u and v have dependency d if $K(u) + K(v) - K(uv) \leq d(\max(|u|, |v|))$; we say that two sequences x and y have dependency d if, for every n and m sufficiently large, the strings $x\restriction n$ and $y\restriction m$ have dependency d. With this terminology, sequences x and y are finitary-independent if they have dependency $c \cdot \log n$, for some positive constant c.

The question becomes: How large can d be so that an effective increase of the randomness rate is possible from two sequences with dependency d? Miller's result shows that this is impossible for dependency $d(n) = n$, while the result in [Zim08] shows that this is possible for dependency $d(n) = c \cdot \log n$. In fact, [Zim08] shows that, for certain combinations of parameters, an effective increase is possible even for dependency $d(n) = n^\alpha$, for some $0 < \alpha < 1$. More precisely, it is shown that for any $\tau > 0$ and $\delta > 0$, there exists $0 < \alpha < 1$ and a truth-table reduction f such that for any sequences x and y that have randomness rate τ and dependency $d(n) = n^\alpha$, it holds that $f(x, y)$ has randomness rate $1 - \delta$.

[1] In [Zim08], such sequences are called independent. The paper [CZ08] examines thoroughly the concept of algorithmic independence for sequences and introduces besides finitary-independence, a stronger concept which is called independence. We adopt here the terminology from [CZ08].

In this paper, we improve the above result and show that one can effectively increase the randomness rate even for two input sources that have linear dependency. More formally, our result is:

(1) We show that for every $0 < \tau \leq 1$ and $\delta > 0$, there exist $0 < \alpha < 1$ and a truth-table reduction f such that for any sequences x and y with randomness rate τ and dependency $d(n) = \alpha n$, the sequence $f(x, y)$ has randomness rate $\geq (1 - \delta)$.

We also study the finite version of the problem, when the input consists of strings. Similarly to the infinite case, our interest is in determining how many input strings and what level of dependency are necessary in order to exist an effective procedure that extracts Kolmogorov complexity. Vereshchagin and Vyugin [VV02, Th. 4] have shown that one input string is not enough. They construct a string x so that any shorter string that has small Kolmogorov complexity conditioned by x (in particular any string effectively constructed from x) has small Kolmogorov complexity unconditionally. On the other hand, Fortnow, Hitchcock, Pavan, Vinodchandran and Wang [FHP$^+$06] show that an input consisting of several independent strings can accomplish the task, when the number of strings in the input depends on the complexity of the strings. Formally, they show that, for any σ there exists a constant ℓ and a polynomial-time procedure that from an input consisting of ℓ n-bit strings $x_1, \ldots, x_\ell$, each with Kolmogorov complexity at least σn, constructs an n-bit string with Kolmogorov complexity $\succeq n - \mathrm{dep}(x_1, \ldots, x_\ell)$ ($\mathrm{dep}(x_1, \ldots, x_\ell) = \sum_{i=1}^{\ell} K(x_i) - K(x_1 \ldots x_\ell)$ and $\succeq$ means that the inequality holds within an error of $O(\log n)$). In view of Vereshchagin-Vyugin result, the question is whether effective extraction of Kolmogorov complexity is possible from two input strings. We have two results in this regard:

(2) We show that if strings x and y of length n have dependency αn and complexity σn, then it is possible to effectively construct a string of length $\approx 2\sigma \cdot n$ and complexity $\succeq (2\sigma - \alpha) \cdot n$, where $\approx$ ($\succeq$) means that the equality (resp., the inequality) is within an error of $O(\log n)$. The construction is uniform in x, y, α, σ. Note, however, that unlike the procedure from [FHP$^+$06], the construction does not run in polynomial time.

(3) Our second result shows that from strings x and y, with sufficiently large complexity and sufficiently small dependency, it is possible to construct a string z that has large complexity even conditioned by any of the input strings. More precisely if x and y are strings of length n that have complexity $s(n)$ and dependency $\alpha(n)$, then it is possible to effectively construct a string of length $m \approx s(n)/2$ such that $K(z \mid x) \succeq m - \alpha(n)$ and $K(z \mid y) \succeq m - \alpha(n)$. The construction is uniform in $x, y, s(n), \alpha(n)$. This improves a result from [CZ08], where the input consists of three strings x_1, x_2, x_3 and the construction produces a string z with large $K(z \mid x_i)$, $i = 1, 2, 3$.

Effective procedures that extract the Kolmogorov complexity of strings are related to randomness extractors. These are objects of major interest in computational complexity and there is a long and very active line of research dedicated to them. A randomness extractor is a procedure (which, ideally, runs in polynomial time) that improves the quality of a defective source of randomness. A source of randomness is modeled by a distribution X on $\{0,1\}^n$, for some n, and its quality is modeled by the min-entropy of X (X has min-entropy k if 2^{-k} is the largest probability that X assigns to any string in $\{0,1\}^n$). The distribution X is defective if its min-entropy is less than n, and is perfect if its min-entropy is equal to n, which implies that X is the uniform distribution on $\{0,1\}^n$. In many applications, it is

desirable to transform a defective distribution X into a distribution X' on a set of shorter strings which is close to the uniform distribution. Such a transformation is called a randomness extractor. Randomness extraction is not possible from a single source [SV86], but it is possible from two or more sources [Vaz87]. Consequently, the research has focused on two types of extractors, seeded extractors and multi-source extractors. A seeded extractor extracts randomness from two independent distributions X and Y, where X is defective and defined on the set of n-bit strings and Y is perfect and defined on the set of d-bit strings, with d much shorter than n (typically $d = O(\log n)$). A k-multisource extractor takes as input k defective distributions on the set of n-bit strings. For $k = 2$, the best multisource extractors are (a) the extractor given by Raz [Raz05] with one source having min-entropy $((1/2) + \alpha)n$ (for some small α) and the second source having min-entropy polylog(n), and (b) the extractor given by Bourgain [Bou05] with both sources having min-entropy $((1/2) - \alpha)n$ (for some small α). There is a clear analogy between randomness extractors and procedures that extract Kolmogorov complexity. In particular, the reader may compare results (2) and (3) discussed above with existing 2-multisource extractors, but we emphasize that there is a major difference in that extractors run in polynomial time, while the procedures in (2) and (3) are only in EXPSPACE. On the other hand, results (2) and (3) suggest that it might be possible to construct multisource extractors with sources having a certain level of dependence and/or with the output being random even conditioned by one of the sources.

A few words about the proof technique. At the highest level, our method follows the structure of the proofs in [Zim08]. One key idea is taken from Fortnow et al. [FHP$^+$06], who showed that a multisource extractor also extracts Kolmogorov complexity. Since multisource extractors with the parameters that are needed here are not known to exist, we construct a combinatorial object, called a balanced table, that is similar with a 2-multisource extractor. A balanced table is a 2-dimensional $N \times N$ table with each entry having one of M colors such that in each sufficiently large subrectangle all the colors appear approximately the same number of times (see Definition 2.2). Using the probabilistic method, we show the existence of balanced tables with appropriate parameters. It follows that such tables can be effectively constructed using exhaustive search. Next, using arguments similiar to those in [FHP$^+$06], we show that if x and y have sufficiently large complexity and sufficiently small dependence, then the color of the entry in row x and column y of the table has large complexity. These ideas are sufficient to establish result (2) (Theorem 3.1). Results (1) (Theorem 4.1) and (3) (Theorem 3.2) require non-trivial technical refinements of the basic method which are explained in the respective proofs.

2. Preliminaries

2.1. Notation

We work over the binary alphabet $\{0, 1\}$. A string is an element of $\{0, 1\}^*$ and a sequence is an element of $\{0, 1\}^\infty$. If x is a string, $|x|$ denotes its length. If x is a string or a sequence and $n \in \mathbb{N}$, $x\restriction n$ denotes the prefix of x of length n. The cardinality of a finite set A is denoted $|A|$. For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, \ldots, n\}$. Let M be a standard Turing machine. For any string x, define the *(plain) Kolmogorov complexity* of x with respect to M, as

$$K_M(x) = \min\{|p| \mid M(p) = x\}.$$

There is a universal Turing machine U such that for every machine M there is a constant c such that for all x,

$$K_U(x) \leq K_M(x) + c. \tag{2.1}$$

We fix such a universal machine U and dropping the subscript, we let $K(x)$ denote the Kolmogorov complexity of x with respect to U. For the concept of conditional Komogorov complexity, the underlying machine is a Turing machine that in addition to the read/work tape which in the initial state contains the input p, has a second tape containing initially a string y, which is called the conditioning information. Given such a machine M, we define the Kolmogorov complexity of x conditioned by y with respect to M as

$$K_M(x \mid y) = \min\{|p| \mid M(p, y) = x\}.$$

Similarly to the above, there exist universal machines of this type and they satisfy the relation similar to Equation 2.1, but for conditional complexity. We fix such a universal machine U, and dropping the subscript U, we let $K(x \mid y)$ denote the Kolmogorov complexity of x conditioned by y with respect to U.

Let $\sigma \in [0, 1]$. A sequence x has randomness rate at least σ if $K(x(1 : n)) \geq \sigma \cdot n$, for almost every n (i.e., the set of n's violating the inequality is finite).

The procedures that we design for extracting the Kolmogorov complexity of strings or sequences are either computable functions (in the case of strings) or Turing reductions (in the case of sequences). In our result, the Turing reduction is also uniform in two parameters τ and σ. Formally, such a Turing reduction f is represented by a two-oracle Turing machine M_f. The machine M_f has access to two oracles x and y, which are binary sequences. When M_f makes the query "n-th bit of first oracle?" ("n-th bit of second oracle?"), the machine obtains $x(n)$ (respectively, $y(n)$). On input $(\tau, \sigma, 1^n)$, where τ and σ are rational numbers (given in some canonical representation), M_f outputs one bit. We say that $f(x, y, \tau, \sigma,) = z \in \{0, 1\}^\infty$, if for all n, M_f on input $(\tau, \sigma, 1^n)$ and working with oracles x and y halts and outputs $z(n)$. In case the machine M_f halts on all inputs and with all oracles, we say that f is a truth-table reduction.

2.2. Limited Independence

Definition 2.1. (a) The dependency of two strings x and y is $\mathrm{dep}(x, y) = K(x) + K(y) - K(xy)$.

(b) Let $d : \mathbb{N} \to \mathbb{R}$. We say that strings x and y have dependency at most $d(n)$ if $\mathrm{dep}(x, y) \leq d(\max(|x|, |y|))$.

(c) Let $d : \mathbb{N} \to \mathbb{R}$. We say that sequences x and y have dependency at most $d(n)$, if for every natural numbers n and m, the strings $x\restriction n$ and $y\restriction m$ have dependency at most $d(n)$.

2.3. Balanced Tables

Let N and M be positive integers. An (N, M) table is a function $T : [N] \times [N] \to [M]$. It is convenient to view it as a two dimensional table with N rows and N columns where each entry has a color from the set $[M]$. If B_1, B_2 are subsets of $[N]$, the $B_1 \times B_2$ rectangle of table T is the part of T comprised of the rows in B_1 and the columns in B_2.

Definition 2.2. Let $T : [N] \times [N] \to [M]$ be an (N, M) table and $S \leq N$ and $D \leq M$ be two positive integers. We say that the table is (S, D)-balanced if for every set $A \subseteq [M]$ with $|A| = M/D$ and for every sets $B_1 \subseteq [N], B_2 \subseteq [N]$ with $|B_1| \geq S, |B_2| \geq S$,

$$|T^{-1}(A) \cap (B_1 \times B_2)| \leq 2 \cdot \frac{|A|}{M} \cdot |B_1 \times B_2|.$$

The above definition states that in any $B_1 \times B_2$ rectangle of T and for any set A of colors of size M/D, the fraction of occurences of colors in A is bounded by $2 \cdot |A|/M$.

Lemma 2.3. *Suppose $S^2 > 3M + 3M \ln D + 6SD + 6SD \ln(N/S)$. Then there exists a table $T : [N] \times [N] \to [M]$ that is (S, D)-balanced.*

Proof. The proof is by the probabilistic method. We color the N-by-N table selecting for each entry independently at random a color from $[M]$. Let us fix $A \subseteq [M]$ with $|A| = M/D$, $B_1 \subseteq [N]$ with $|B_1| = S$ and $B_2 \subseteq [N]$ with $|B_2| = S$. Note that it is enough to prove the assertion for sets B_1 and B_2 of size exactly S. By the Chernoff bounds,

$$\text{Prob}\left(\frac{\text{number of } A\text{-colored cells in } B_1 \times B_2}{S^2} > 2\frac{|A|}{M}\right) \leq e^{-(1/3)(|A|/M)S^2} = e^{-(1/(3D))S^2}. \tag{2.2}$$

The number of possibilities of choosing the set A as above is bounded by

$$\binom{M}{M/D} \leq (e \cdot D)^{M/D} = e^{M/D + (M/D)\ln D}. \tag{2.3}$$

The number of possibilities of choosing the sets B_1 and B_2 as above is bounded by

$$\binom{N}{S}^2 \leq (eN/S)^{2S} = e^{2S + 2S \cdot \ln(N/S)}. \tag{2.4}$$

The hypothesis ensures that the product of the upper bounds in Equations (2.2), (2.3), and (2.4) is less than 1. It follows from the union bound that there exists an (S, D)-balanced table. ∎

In our applications, N and M will be powers of two, $N = 2^n$, $M = 2^m$, and $[N]$ is identified with $\{0,1\}^n$ and $[M]$ is identified with $\{0,1\}^m$. We assume this setting in the following.

Lemma 2.4. *Let $T : [N] \times [N] \to [M]$ be an (S, M)-balanced table. Let v be a string with $|v| \leq m$. Then for all sets $B_1 \subseteq [N], B_2 \subseteq [N]$ with $|B_1| \geq S, |B_2| \geq S$, the number of entries in the $B_1 \times B_2$ rectangle of T that have a color whose prefix is v is $\leq 2 \cdot \frac{1}{2^{|v|}} \cdot |B_1 \times B_2|$.*

Proof. First observe that, since the table is (S, D)-balanced with the value of the parameter D equal to M, the definition of an (S, D)-balanced table implies that no color $a \in [M]$ occurs more than a fraction of $2/M$ times in any rectangle of T with sizes $\geq S$. Let v be a string of length of most m. Then v has $2^{m-|v|}$ extensions of length m and, as we have just noted, each such extension occurs at most a fraction $2/M$ in any rectangle with sizes $\geq S$. It follows that in any $B_1 \times B_2$ rectangle of T, all the extensions of v taken together occur at most $2^{m-|v|} \cdot (2/M) \cdot |B_1 \times B_2| = (2/2^{|v|}) \cdot |B_1 \times B_2|$ times. ∎

3. Increasing the randomness rate of strings

The next theorem shows that from two n-bit strings with complexity σn and dependency αn, one can construct a string of length $\approx 2\sigma n$ and complexity $\approx (2\sigma - \alpha)n$.

Theorem 3.1. *For every $\sigma > 0$, for every $0 < \alpha < \sigma$, there is a computable function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$, that, for every n, maps any pair of strings of length n into a string of length $m = 2\sigma n - \log n$ and has the following property: for every sufficiently large n, if (x, y) is a pair of strings with*

(1) $|x| = |y| = n$,
(2) $K(x) \geq \sigma n$, $K(y) \geq \sigma n$
(3) (x, y) *have dependency at most αn,*

then

$$K(f(x, y)) \geq (2\sigma - \alpha)n - 9 \log n.$$

Proof. Let us fix n and let $N = 2^n, m = 2\sigma n - \log n, M = 2^m, S = 2^{\sigma n}, d = \alpha n + 8 \log n$, and $D = 2^d$. Note that the requirements of Lemma 2.3 are satisfied and therefore there exists a table $T : [N] \times [N] \to [M]$ that is (S, D)-balanced. By brute force, we find the smallest (in some canonical sense) such table T. Note that the table T can be described with $\log n + O(1)$ bits. We define $f(x, y)$ to be $T(x, y)$. Thus, let $z = T(x, y)$ for some strings x and y of length n satisfying the requirements in the theorem hypothesis. For the sake of obtaining a contradiction, suppose that $K(z) < (2\sigma - \alpha)n - 9 \log n = m - d$. Let $t_1 = K(x), t_2 = K(y)$. From the properties of x and y, $t_1 \geq \sigma n$ and $t_2 \geq \sigma n$. Let $B_1 = \{u \in \{0,1\}^n \mid K(u) \leq t_1\}$, $B_2 = \{v \in \{0,1\}^n \mid K(v) \leq t_2\}$ and $A = \{w \in \{0,1\}^m \mid K(w) < m - d\}$. We have $|B_1| \leq 2^{t_1+1}$, $|B_2| \leq 2^{t_2+1}$ and $|A| < 2^{m-d}$. We take B_1' and B_2' with $|B_1'| = 2^{t_1+1}, |B_2'| = 2^{t_2+1}$, $B_1 \subseteq B_1'$ and $B_2 \subseteq B_2'$. Since the table T is (S, D)-balanced,

$$\begin{aligned} |T^{-1}(A) \cap (B_1 \times B_2)| \leq |T^{-1}(A) \cap (B_1' \times B_2')| &\leq 2 \cdot \tfrac{|A|}{M} \cdot |B_1' \times B_2'| \\ &\leq 2 \cdot 2^{m-d} \tfrac{1}{2^m} \cdot 2^{t_1+1} \cdot 2^{t_2+1} \\ &\leq 2^{t_1+t_2-d+3}. \end{aligned}$$

Note that $(x, y) \in T^{-1}(A) \cap (B_1 \times B_2)$ and that $T^{-1}(A) \cap (B_1 \times B_2)$ can be enumerated if we are given t_1, t_2 and n (from which $(m - d)$ and a description of table T can be determined). Therefore xy can be described by the rank of (x, y) in the above enumeration and by information needed for performing that enumeration. Thus

$$\begin{aligned} K(xy) &\leq t_1 + t_2 - d + 2(\log t_1 + \log t_2 + \log n) + O(1) \\ &\leq t_1 + t_2 - d + 7 \log n. \end{aligned}$$

For the second inequality, we took into consideration that $t_1 \leq n$ and $t_2 \leq n$. On the other hand, since x and y have dependency bounded by αn.

$$K(xy) \geq t_1 + t_2 - \alpha n.$$

Keeping in mind that $d = \alpha n + 8 \log n$, we have obtained a contradiction. ∎

The next theorem shows from two n-bit strings with complexity $s(n)$ and dependency $\alpha(n)$, one can construct a string of length $m \approx s(n)/2$ with complexity conditioned by any one of the input strings $\approx m - \alpha(n)$.

Theorem 3.2. *For every computable function $s(n)$ verifying $6 \log n < s(n) \leq n$ and every function $\alpha(n)$, there is a computable function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ that, for every*

n, maps any pair of strings of length n into a string of length $m = s(n)/2 - 7\log n$ and has the following property: for every sufficiently large n, if (x, y) is a pair of strings with

(1) $|x| = |y| = n$,
(2) $K(x) \geq s(n), K(y) \geq s(n)$,
(3) (x, y) *has dependency at most* $\alpha(n)$

then

$$\begin{aligned} K(f(x,y) \mid x) &\geq m - \alpha(n) - 11\log n, \\ K(f(x,y) \mid y) &\geq m - \alpha(n) - 11\log n. \end{aligned}$$

Proof. We fix n and let $N = 2^n, m = s(n)/2 - 7\log n, M = 2^m, S = 2^{s(n)/2}, D = M, t = \alpha(n) + 11\log n$. The requirements of Lemma 2.3 are satisfied and therefore there exists a table $T : [N] \times [N] \to [M]$ that is (S, D)-balanced. By brute force, we find the smallest (in some canonical sense) such table T. The table T is determined by n and $s(n)$, and, thus, can be described with $\log n + \log s(n) + O(1)$ bits. Note that, since $D = M$, it holds that for every color $a \in [M]$ and for every subsets $B_1 \subseteq [N], B_2 \subseteq [N]$ with $|B_1| \geq S, |B_2| \geq S$, the number of occurrences of a in the $B_1 \times B_2$ subrectangle of T is bounded by $(2/M) \cdot |B_1 \times B_2|$.

We define $f(x, y)$ to be $T(x, y)$. Thus, let $z = T(x, y)$ for some strings x and y of length n satisfying the requirements in the theorem hypothesis. We need to show that $K(z \mid x)$ and $K(z \mid y)$ are at least $m - \alpha(n) - 11\log n$. We show this relation for $K(z \mid y)$ (the proof for $K(z \mid x)$ is similar). For the sake of obtaining a contradiction, suppose that $K(z \mid y) < m - \alpha(n) - 11\log n = m - t$. Let $t_1 = K(x)$. Note that $t_1 \geq s(n)$. Let $B = \{u \in \{0,1\}^n \mid K(u) \leq t_1\}$. Note that $2^{t_1+1} > |B| \geq 2^{s(n)/2} = S$. We say that a column $u \in [N]$ is *bad for color* $a \in [M]$ *and* B if the number of occurrences of a in the $B \times \{u\}$ subrectangle of T is greater that $(2/M) \cdot |B|$ and we say that u is *bad for* B if it is bad for some color a and B. For every $a \in [M]$, the number of u's that are bad for a and B is $< S$ (because T is (S, D)-balanced). Therefore, the number of u's that are bad for B is $< M \cdot S$. Given t_1 and a description of the table T, one can enumerate the set of u's that are bad for B. This implies that any u that is bad for B can be described by its rank in this enumeration and the information needed to perform the enumeration. Therefore, if u is bad for B,

$$\begin{aligned} K(u) &\leq \log(M \cdot S) + 2(\log t_1 + \log n + \log s(n)) + O(1) \\ &\leq m + s(n)/2 + 6\log n + O(1) \\ &< s(n), \end{aligned}$$

provided n is large enough. Since $K(y) \geq s(n)$, it follows that y is good for B.

Let $A = \{w \in [M] \mid K(w \mid y) < m - t\}$. We have $|A| < 2^{m-t}$ and, by our assumption, $z \in A$. Let G be the subset of B of positions in the strip $B \times \{y\}$ of T having a color from A (formally, $G = \mathrm{proj}_1(T^{-1}(A) \cap (B \times \{y\}))$. Note that x is in G. Each color a occurs in the strip $B \times \{y\}$ at most $(2/M) \cdot |B|$ (because y is good for B). Therefore the size of G is bounded by

$$|A| \cdot (2/M) \cdot |B| \leq 2^{m-t} \cdot (2/M) \cdot 2^{t_1+1} < 2^{t_1-t+2}.$$

Given $y, t_1, m-t$ and a description of the table T, one can enumerate the set G. Therefore, x can be described by its rank in this enumeration and by the information needed to perform

the enumeration. It follows that

$$\begin{aligned} K(x \mid y) &\le t_1 - t + 2 + 2(\log t_1 + \log(m-t) + \log n + \log s(n)) + O(1) \\ &\le t_1 - t + 8\log n + O(1) \\ &= t_1 - \alpha(n) - 3\log n + O(1) \\ &= K(x) - \alpha(n) - 3\log n + O(1). \end{aligned}$$

Since $K(xy) \le K(y) + K(x \mid y) + 2\log n + O(1)$ (this holds for every n-bit strings x and y), we obtain

$$\begin{aligned} K(xy) &\le K(y) + K(x) - \alpha(n) - 3\log n + 2\log n + O(1) \\ &\le K(y) + K(x) - \alpha(n) - \log n + O(1), \end{aligned}$$

which contradicts that x and y have dependency at most $\alpha(n)$. ∎

4. Increasing the randomness rate of sequences

We prove that the randomness rate of sequences can be effectively increased even from two sequences having linear dependence.

Theorem 4.1. *There exists a truth-table reduction f with the following property. For any rational numbers $\tau > 0$ and $\delta > 0$, there exists $\alpha > 0$ such that for any sequences x and y with randomness rate at least τ and dependency at most αn, $f(x, y, \tau, \delta)$ has randomness rate at least $1 - \delta$. Moreover, the reduction f is uniform in x, y, τ and δ.*

Proof. The plan is as follows. We split x into strings $x_1 x_2 \dots x_i \dots$ and y into strings $y_1 y_2 \dots y_i \dots$. For each i, let $\overline{x}_i = x_1 \dots x_i$ and $\overline{y}_i = y_1 \dots y_i$. The splitting is done in such a way that x_i and y_i have complexity close to $\tau|x_i|$ and respectively close to $\tau|y_i|$ even conditioned by $\overline{x}_{i-1}\overline{y}_{i-1}$. Next, for each i, we construct a balanced table T_i with appropriate parameters and take $z_i = T(x_i, y_i)$. The output of the truth-table reduction is the sequence $z = z_1 z_2 \dots z_i \dots$. As in the case of strings, it follows that z_i has high complexity and actually this holds even conditioned by $\overline{z}_{i-1} = z_1 z_2 \dots z_{i-1}$. So far, the proof is as in [Zim08]. The point of departure is that in order for the construction to work with inputs having linear dependence, we need to take the length of z_i exponential in i (rather than quadratic in i, which was the case in [Zim08]). This creates difficulties in showing that every "intermediate" prefix of z (i.e., a string that is an extension of $\overline{z}_{i-1}$ and a prefix of $\overline{z}_i$, for some i) has high complexity. To handle this, we argue that even prefixes of z_i have relatively high Kolmogorov complexity conditioned by $\overline{x}_{i-1}\overline{y}_{i-1}$ (see Lemma 4.3) and then the argument for "intermediate" strings forks into two cases depending on whether the string is long or short (see Lemma 4.4).

We proceed with the formal proof.

We fix rational numbers $\tau > 0$ and $\delta > 0$. Let x and y be sequences with randomness rate at least τ. Let $\epsilon = \delta/4$.

We split $x = x_1 x_2 \dots x_i \dots$ and $y = y_1 y_2 \dots y_i \dots$ and let $n_i = |x_i| = |y_i|$. We'll take $n_i = B^i$ for some constant B, given by the next lemma.

Lemma 4.2. *There exists a constant $B > 1$ with the following properties:*

(a) *For every i, $K(x_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) \ge 0.99\tau n_i$ and $K(y_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) \ge 0.99\tau n_i$.*

(b) *For any $\alpha > 0$, if (x, y) have dependency αn, then, for all i*

$$K(x_i y_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) \ge K(x_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) + K(y_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) - (2.1) \cdot \alpha \cdot n_i.$$

Proof. (Sketch.) The proof is similar to an analogous result from [Zim08]. For (a), it is easy to show that B can be taken large enough so that the length of x_i is so much larger than the length of $\overline{x}_{i-1}\overline{y}_{i-1}$ that the complexity of x_i does not decrease too much if it is conditioned by $\overline{x}_{i-1}\overline{y}_{i-1}$.

The proof of (b) passes through the following intermediate steps:

(1) We show that for B, i and j sufficiently large,

$$K(\overline{y}_i\overline{x}_j) = K(\overline{y}_i) + K(\overline{x}_j) \pm 1.001\alpha(B^i + B^j).$$

(This is the analogue of Lemma 4.5 from [Zim08]).

(2) We show that B, i and j sufficiently large,

$$K(x_i \mid \overline{x}_{i-1}\overline{y}_j) = K(x_i \mid \overline{x}_{i-1}) \pm 2.004\alpha(B^i + B^j).$$

(This is the analogue of Lemma 4.6 from [Zim08]; the constants are not optimized).

Next, the statement can be shown similarly to Lemma 4.7 from [Zim08]). ∎

For the rest of this section, we fix the following parameters as follows:

- The constant B is as given by Lemma 4.2,
- $\alpha = (1/3)\epsilon^2 \cdot (0.97\tau) \cdot (1/B)$.
- For each i, $N_i = 2^{n_i}$, $S_i = 2^{(0.98\tau)\cdot n_i}$, $m_i = (0.97\tau) \cdot n_i$, $M_i = 2^{m_i}$ $D_i = M_i$.

The parameters satisfy the requirements of Lemma 2.3 and, thus, for each i, there exists a table $T_i : [N_i] \times [N_i] \to [M_i]$ that is (S_i, D_i)-balanced. For every i, given i, a smallest (in some canonical sense) such table T_i can be constructed by exhaustive search. We fix these tables T_i and define $z_i = T_i(x_i, y_i)$ and next $z = z_1 z_2 \dots z_i \dots$. Clearly z is constructed by a truth-table reduction f from input sequences x and y. We will show that z has randomness rate at least $1 - \delta$.

Lemma 4.3. *For every i sufficiently large, each prefix v of z_i has $K(v \mid \overline{x}_{i-1}\overline{y}_{i-1}) \geq |v| - 3\alpha \cdot n_i$.*

Proof. Suppose that there is a prefix v of z_i with $K(v \mid \overline{x}_{i-1}\overline{y}_{i-1}) < |v| - 3\alpha \cdot n_i$. We define:

- $t_1 = K(x_i \mid \overline{x}_{i-1}\overline{y}_{i-1})$, $t_2 = K(y_i \mid \overline{x}_{i-1}\overline{y}_{i-1})$,
- $B_1 = \{u \in \{0,1\}^{n_i} \mid K(u \mid \overline{x}_{i-1}\overline{y}_{i-1}) \leq t_1\}$,
- $B_2 = \{u \in \{0,1\}^{n_i} \mid K(u \mid \overline{x}_{i-1}\overline{y}_{i-1}) \leq t_2\}$,
- $A = \{w \in \{0,1\}^{|v|} \mid K(w \mid \overline{x}_{i-1}\overline{y}_{i-1}) < |v| - 3\alpha \cdot n_i\}$.

Note that $t_1 \geq 0.99\tau \cdot n_i$, $t_2 \geq 0.99\tau \cdot n_i$ (by Lemma 4.2), $2^{t_1+1} > |B_1| \geq 2^{0.98\tau\cdot n_i} = S_i$, $2^{t_2+1} > |B_2| \geq 2^{0.98\tau\cdot n_i} = S_i$ and $|A| < 2^{|v|-3\alpha n_i}$. Let G be the set of entries (represented by their coordinates in the table) in the $B_1 \times B_2$ rectangle of the table T_i that have a color with a prefix in A. By Lemma 2.4, the cardinality of G is at most

$$\begin{aligned} 2 \cdot \tfrac{|A|}{2^{|v|}} \cdot |B_1 \times B_2| &\leq 2 \cdot 2^{|v|-3\alpha n_i} \cdot \tfrac{1}{2^{|v|}} \cdot 2^{t_1+1} \cdot 2^{t_2+1} \\ &= 2^{t_1+t_2-3\alpha n_i+3}. \end{aligned}$$

Note that (x_i, y_i) belongs to G and that G can be enumerated given $\overline{x}_{i-1}\overline{y}_{i-1}$, t_1, t_2, and $|v| - 3\alpha \cdot n_i$ (observe that i can be determined from $\overline{x}_{i-1}\overline{y}_{i-1}$ and thus the table T_i can be constructed). Therefore $x_i y_i$ can be described by its rank in the enumeration of G and by the information needed to perform this enumeration. This implies

$$\begin{aligned} K(x_i y_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) &\leq t_1 + t_2 - 3\alpha \cdot n_i + 2(\log t_1 + \log t_2 + \log(|v| - 3\alpha n_i)) + O(1) \\ &\leq t_1 + t_2 - 3\alpha \cdot n_i + O(\log n_i). \end{aligned}$$

On the other hand, by Lemma 4.2,

$$K(x_i y_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) \geq K(x_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) + K(y_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) - (2.1)\cdot\alpha\cdot n_i.$$

If i is large, the last two inequalities conflict each other and we obtain a contradiction. ∎

The next lemma finishes the proof of Theorem 4.1.

Lemma 4.4. *For each sufficiently long prefix w of z, $K(w) \geq (1-4\epsilon)|w|$.*

Proof. For some i, the prefix w is of the form $w = z_1 \ldots z_{i-1}v_i$, with v_i a prefix of z_i. Let $\gamma = (1/\epsilon)\cdot(3\alpha)$. We consider two cases:

Case 1: v_i is long. Suppose $|v_i| \geq \gamma \cdot n_i$.

Then $K(v_i \mid \overline{x}_{i-1}\overline{y}_{i-1}) \geq |v_i| - 3\alpha\cdot n_i \geq |v_i| - (3\alpha/\gamma)\cdot|v_i| = (1-\epsilon)|v_i|$. This implies $K(v_i \mid z_1 \ldots z_{i-1}) > (1-\epsilon)\cdot|v_i| - O(1) \geq (1-2\epsilon)|v_i|$, because each z_j can be constructed from x_j and y_j. By induction, it follows that $K(z_1 z_2 \ldots z_{i-1}v_i) \geq (1-3\epsilon)|z_1 z_2 \ldots z_{i-1}v_i|$. For the induction step, the argument goes as follows:

$$\begin{aligned} K(z_1 z_2 \ldots z_{i-1}v_i) &\geq K(z_1 \ldots z_{i-1}) + K(v_i \mid z_1 \ldots z_{i-1}) \\ &\quad -O(\log(m_1 + \ldots + m_{i-1}) + \log(|v_i|)) \\ &\geq (1-3\epsilon)(m_1 + \ldots + m_{i-1}) + (1-2\epsilon)|v_i| \\ &\quad -O(\log(m_1 + \ldots + m_{i-1}) + \log|v_i|)) \\ &> (1-3\epsilon)(m_1 + \ldots + m_{i-1} + |v_i|). \end{aligned}$$

In the last step, we have used the fact that $\log(m_1 + \ldots + m_{i-1}) = O(i), \log|v_i| = O(i)$ and $|v_i| = \Omega(B^i)$.

Case 2: v_i is short. Suppose $|v_i| < \gamma \cdot n_i$.

For a contradiction, suppose $K(z_1 z_2 \ldots z_{i-1}v_i) < (1-4\epsilon)|z_1 z_2 \ldots z_{i-1}v_i|$. Note that $z_1 z_2 \ldots z_{i-1}$ can be reconstructed from a descriptor of $z_1 z_2 \ldots z_{i-1}v_i$. This implies

$$\begin{aligned} K(z_1 z_2 \ldots z_{i-1}) &< (1-4\epsilon)(m_1 + m_2 + \ldots + m_{i-1} + |v_i|) + O(1) \\ &= (1-4\epsilon)(m_1 + \ldots + m_{i-1}) + (1-4\epsilon)|v_i| + O(1) \\ &\leq (1-4\epsilon)(m_1 + \ldots + m_{i-1}) + (1-4\epsilon)\gamma\cdot n_i \\ &\leq (1-4\epsilon)(m_1 + \ldots + m_{i-1}) + (1-4\epsilon)\cdot(1/\epsilon)(3\alpha)\cdot n_i. \end{aligned}$$

But the second term is less than $\epsilon(m_1 + \ldots + m_{i-1})$ (due to the choice of α). This implies that $K(z_1 z_2 \ldots z_{i-1}) \leq (1-3\epsilon)(m_1 + m_2 + \ldots + m_{i-1})$, which, by Case 1, is not possible. ∎

∎

Note. It remains an open issue whether from input sequences x and y (even independent) one can construct a sequence z that has high randomness rate conditioned by any one of the input sequences. In other words, the infinite analogue of Theorem 3.2 is open.

References

[BDS07] L. Bienvenu, D. Doty, and F. Stephan. Constructive dimension and weak truth-table degrees. In *Computation and Logic in the Real World - Third Conference of Computability in Europe*, pages 63–72. Springer-Verlag *Lecture Notes in Computer Science #4497*, 2007. Available as Technical Report arXiv:cs/0701089 at arxiv.org.

[Bou05] J. Bourgain. More on the sum-product phenomemnon in prime fields and its applications. *International Journal of Number Theory*, 1:1–32, 2005.

[CZ08] Cristian S. Calude and Marius Zimand. Algorithmically independent sequences. In Masami Ito and Masafumi Toyama, editors, *Developments in Language Theory*, volume 5257 of *Lecture Notes in Computer Science*, pages 183–195. Springer, 2008.

[FHP+06] L. Fortnow, J. Hitchcock, A. Pavan, N.V. Vinodchandran, and F. Wang. Extracting Kolmogorov complexity with applications to dimension zero-one laws. In *Proceedings of the 33rd International Colloquium on Automata, Languages, and Programming*, pages 335–345, Berlin, 2006. Springer-Verlag *Lecture Notes in Computer Science #4051*.

[Mil08] J. Miller. Extracting information is hard, May 2008. Manuscript, http://www.math.uconn.edu/ josephmiller/Papers/dimension.pdf.

[NR06] A. Nies and J. Reimann. A lower cone in the wtt degrees of non-integral effective dimension. In *Proceedings of IMS workshop on Computational Prospects of Infinity*, Singapore, 2006. To appear.

[Raz05] Ran Raz. Extractors with weak random seeds. In Harold N. Gabow and Ronald Fagin, editors, *STOC*, pages 11–20. ACM, 2005.

[Rei04] J. Reimann. Computability and fractal dimension. Technical report, Universität Heidelberg, 2004. Ph.D. thesis.

[SV86] M. Santha and U. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33:75–87, 1986.

[Vaz87] Umesh V. Vazirani. Strong communication complexity or generating quasirandom sequences from two communicating semi-random sources. *Combinatorica*, 7(4):375–392, 1987.

[VV02] Nikolai K. Vereshchagin and Michael V. Vyugin. Independent minimum length programs to translate between given strings. *Theor. Comput. Sci.*, 271(1-2):131–143, 2002.

[Zim08] Marius Zimand. Two sources are better than one for increasing the Kolmogorov complexity of infinite sequences. In Edward A. Hirsch, Alexander A. Razborov, Alexei L. Semenov, and Anatol Slissenko, editors, *CSR*, volume 5010 of *Lecture Notes in Computer Science*, pages 326–338. Springer, 2008.